Name: NetID:

Final

CS 2110, 21 December 2021, 2:00pm EST

	1	2	3	4	5	6	7	Total
Question	Name	Loop Invs	Short answer	OO	Graphs	Data structures	Concur-rency	
Max	1	10	21	14	13	31	10	100

- This exam is open-note. Collaboration is strictly forbidden. Do not share any portion of your exam with anyone. Do not discuss the exam with anyone besides the instructors until after grades have been released.
- This exam is designed to take no more than **120 minutes**. If you have SDS accommodations, allocate additional time proportional to this duration (e.g. those granted 50% extra time should plan to spend 180 minutes on the exam). Do not spend more than this nominal time working on the exam problems. It is *your responsibility* to manage your time so that you can print, work for the nominal time, scan, and upload by the deadline while accommodating conflicts and technical hiccups.
- Submit to Gradescope *before* the end of your submission window (as shown on the exam distribution page). Aim to submit early, and contact the instructors immediately if you encounter major technical difficulties.
- **Write your answers in the boxes or spaces provided**. Ambiguous answers will be considered incorrect. You should be able to fit your answers easily into the allocated space.
- In some places we have abbreviated or condensed code to reduce the number of pages that must be printed for the exam. In others, code has been obfuscated to test your understanding of the language. This does not mean that it's good style.
- Policy details and submission instructions are available on Canvas. Good luck!

Academic Integrity Statement: I pledge that I have neither given nor received unauthorized aid on this exam. I will not discuss the exam with anyone in this course who has not yet taken it.

(signature)

1. Name (1 point)

Is your name at the top of this page and your NetID at the top of pages 1..10 correct? If either is incorrect, or if you are missing pages, do not proceed; Contact the instructors immediately! Sign the Academic Integrity Statement before submitting your exam.

2. Loop Invariants (10 points)

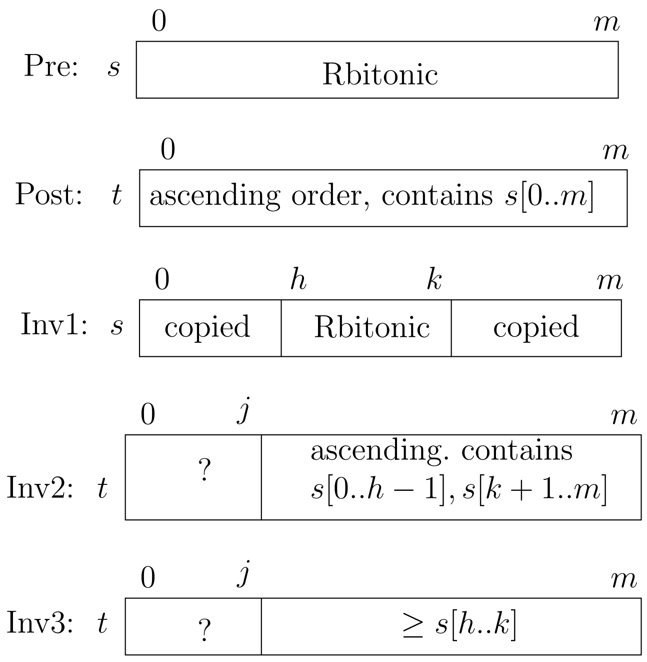
A sequence of integers is *Rbitonic* (reverse bitonic) if it is first descending then ascending. Array s, below, is Rbitonic: The bolded values are descending, the rest are ascending.

	0									m
s	**7**	**6**	**6**	**5**	**3**	**2**	3	3	4	8

We write a while-loop that copies Rbitonic segment $s[0..m]$ into a new array t in ascending order. Example: For the bitonic sequence $s[0..9]$ given above, $t[0..9]$ will be:

	0									m
t	2	3	3	3	4	5	6	6	7	8

(Only part of array s is being re-ordered, not the whole array.) To the right are the precondition, postcondition, and invariant for this problem. The invariant has three parts. Note that values are put into $t[0..m]$ starting at the end, starting with $t[m]$.



(a) 3 points In the box to the right, write the initialization for the while-loop. Do not declare any variables.

```
h= 0;
k= m;
j= m;
```

(b) 2 points The loop has the form **while** (B) *repetend*. To the right, write condition B. Do NOT write anything else.

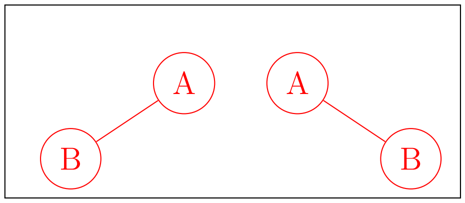
$h \leq k$ or $0 \leq j$ or equivalent

(c) 5 points In the box to the right, write the repetend.

```
if (s[h] > s[k])    // could be >=
      { t[j]= s[h]; h= h + 1; }
else { t[j]= s[k]; k= k − 1; }
j= j - 1;
```

3. Short Answer (21 points)

(a) 3 points. Show that you cannot uniquely construct a binary tree from its preorder and postorder traversals by drawing to the right two *different* binary trees that have the same preorder and postorder traversals.
Hint: Make the trees as small as possible.



(b) 4 points. Consider the following method declaration:

```
void doItPlease(List<Number> nums) {...}
```

Which of the argument types on the right can be passed to method `doItPlease()`? Write the letter for each allowed type in the box below, or write "None" if none are allowed. Note that `Integer` is a subclass of `Number`.

```
T. ArrayList<Number>
U. ArrayList<Integer>
V. LinkedList<Number>
W. List<Integer>
X. Number[]
Y. Object
```

T, V

(c) 4 points. Consider hashing with linear probing using a 7-element array `d[0..6]` to maintain a set of Integers. Initially, `b` contains no elements and is thus: `[null, null, null, null, null, null, null]`. The hash function to be used is the integer itself: `hashcode(i) = i`. Draw array `d` after these values have been added: 18, 6, 13, 7. Do not be concerned with the load factor.

[13, 7, null, null, 18, null, 6]

(d) 6 points. Consider the following snippet of Swing code for changing some text in a window when a button is clicked:

```
JLabel txt = ...;
JButton b = ...;
b.addActionListener(v -> txt.setText("Action: " + v.getActionCommand()));
```

1. What variable or parameter corresponds to the event *source*? b
2. What variable or parameter corresponds to the *event*? v
3. What *interface* does the anonymous function implement? ActionListener

(e) 2 points. Consider an array `d` of objects of a class `Date`. We want to sort array `d` chronologically: sort the `Date` objects by year, but within each year sort by month.
For example, these pairs are in order:
April 2021, May 2021, January 2022.

We do this with the sequence of two statements shown above to the right.

Between quick-sort and merge-sort, only one is appropriate to use here. Which one, and why?

```
Sort array d based on the month,
    e.g. April comes before May;
Sort array d based on the year.
```

Merge-sort – must be stable

(f) 2 points. Consider a graph with n nodes and e edges. Algorithms M and P perform the same task but with different asymptotic complexity. Algorithm M's time complexity is in $O(en^2)$, while P's is in $O(n^3 \lg(n))$. Which algorithm would you expect to run faster for *large, dense* graphs? Write its name (and nothing else) in the box.

P

4. Object-Oriented Programming (14 points)

(a) 7 points You maintain a prolific authors list. You want to keep track of what authors have collaborated, and you want to sort authors using that information.
This will be done using class `Author`. Two of its fields are shown to the right.
For example, `Neil` wrote 5 times with `Terry`, so the pair (object for Terry, 5) occurs in field `wroteWith` in the object for `Neil`.

```
public class Author
        implements Comparable<Author> {
 /** The id of this author */
 public int authorId;

 /** A pair (i, val) is in wroteWith
   * if i has written with this author
   * val times. */
 public HashMap<Author, Integer> wroteWith;
```

Complete the parts of the class indicated by boxes below. Note that many other features of this class are missing.

```
/** Constructor: An instance with authorId id that hasn't written with anyone else */
public Author(int id) {
```

authorId= id;
wroteWith= new HashMap<Author, Integer>();

```
 }

/** = number of times this author has written with every author in wroteWith. */
public int allWrites() {...}

/** Return -1 or 1 depending on whether this author's allWrites()
    * are < or > aut's allWrites(). If this author's allWrites() is
    * equal to aut's allWrites(), return -1, 0, or 1 depending on
    * whether this author's authorId is <, =, or > aut's authorID.
  * Precondition: aut is not null */
public @Override int compareTo(Author aut) {
```

```
    int val= Integer.compare(allWrites(), aut.allWrites());
    if(val == 0) return Integer.compare(authorID, aut.authorID);
    return val;
```

```
}
```

(b) 7 points To the right is part of class `St`, which implements a bounded set in an array. Make it iterable by completing methods `hasNext` and `next` of inner class `SetIterator`, below.

We abbreviate the class invariants and method specs.

Don't introduce any new fields; you can use local variables.

```
class St<E> implements Iterable<E> {
   /** The elements of the set are
      * in d[0..size-1] */
   protected E[] d;
   protected int size; // Size of set.

   public Iterator iterator() {
      return new SetIterator();
   }
```

```
/** An instance is an iterator over the set. */
private class SetIterator implements Iterator {
    /** values  in d[0..h-1] have been enumerated. */
    private int h= 0;

    public SetIterator() {} // Constructor: an iterator over this set

    /** = "there is another value to enumerate." */
    public boolean hasNext() {
```

```
            return h < size;
```

```
    }

    /** Return the next value to enumerate, thus enumerating it.<br>
     * Throw a NoSuchElementException if there is no next element. */
    public E next() {
```

```
        if (!hasNext()) throw new NoSuchElementException();
        E val= d[h];        OR      h= h+1;
        h= h + 1;                   return d[h-1];
        return val;
```

```
 } } }
```

5. Graphs(13 points)

The answer to each of the first 3 questions is one of these algorithms: DFS, BFS, Prim. Write the answer in the box to the right of each question. Don't write anything else in the box.

(a) 2 points Consider a graph with all edge weights 2. Dijkstra's shortest path algorithm to find the distance from one node to all others visits the nodes in the same order as one of the algorithms. Which one?

BFS

(b) 2 points An undirected graph, has cities as nodes and roads between them as edges. Each edge weight is its distance in miles. Which algorithm is best suited to determining whether we can get from a city `C1` to another city `C2` by going through at most one other town?

BFS

(c) 2 points In a third-world country, a road network is being designed to link small outlying villages. An undirected graph has been constructed. Its nodes are the villages. Its edges are possible roads between villages; their weights are the cost to build those roads. What algorithm would you use to determine which roads to build that connect all villages at the minimum cost?

Prim

(d) 7 points To the right are two properties of a spanning tree of an undirected connected graph G. Based on (1), we wrote this abstract algorithm for creating a spanning tree.

(A1) While G has a cycle, throw out one edge of a cycle.

(1) A spanning tree of G is a maximal set of edges that contains no cycle.

(2) A spanning tree of G is a minimal set of edges that connects all nodes.

(d1) Below, give the abstract algorithm A2 that results from using property (2).

Start with all nodes and no edges of G.
While the nodes are not all connected, add an edge that connects two unconnected components. (You can also say "that does not introduce a cycle.")

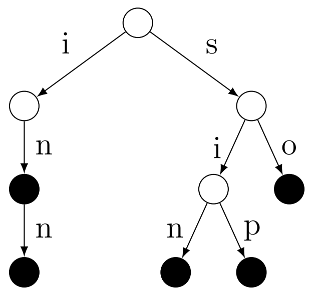
(d2) Each of the abstract algorithms (A1) and (A2) has a loop. Write down the number of iterations each loop takes (as a function of the number n of nodes and the number e of edges).

A1 iterations	A2 iterations
e - (n-1)	n-1

6. Data structures (31 Points)

(a) Data structure for words (15 points)

The tree to the right illustrates a neat data structure for maintaining a set of words in the English language. Assume all letters are in lower case. This tree contains the words "in", "inn", "sin", "sip", and "so". A word in the set is found by reading the characters on a path going down to a black node.



prefix tree

Each node has a 26-element array `z[0..25]` for the children, each of which represents a lowercase letter and is either null or a (pointer to) a child. Each node has a boolean field to say whether it ends a word or not (black or white in the tree to the right).

(a1) 3 points For the children of a node, `z[0]` is for character 'a', `z[1]` for 'b', etc. Let char variable `ch` contain a lowercase letter. In the box fill in the **index** so that the array reference returns the value in `z` corresponding to character `ch`:

z[$\boxed{\text{ch – 'a'}}$]

(a2) 12 points Below, give the tightest expected and worst-case Big-O time complexity bounds, in terms of n and s, for searching for a word of length `n` in a set of Strings of size `s` when the set is implemented as (1) a prefix tree, as above, (2) a balanced Binary Search Tree (BST) of Strings, and (3) a HashSet<String>. Remember: *The cost of determining whether one String equals another, as well as the cost of computing a String's hash code, depends on the length of the String.*

Prefix tree as above. Expected:	O(n)	Worst-case:	O(n)
Balanced BST. Expected:	O(n log s)	Worst-case:	O(n log s)
HashSet<String>. Expected:	O(n)	Worst-case:	O(s*n)

(b) Complexity (10 points) Consider an undirected connected graph with n nodes and e edges. Each node is an object of class `Node`, and the neighbors of a node are given as a `DList<Node>`, where `DList` is your doubly-linked list class from assignment A4, made iterable. In method `viz` on the next page, each statement has a label on it, so we can talk about it.

Consider the call `viz(node, new HashSet<Node>())` of the method below. To the right of each statement in the body of `viz`, write the total number of times it is executed during execution of this call, in terms of n and e. For 'd', write the number of times the for-each statement is encountered, not how many iterations it does.

```
/** Visit all nodes reachable from w along unvisited paths
  * Precondition. v contains all visited nodes. */
static void viz(Node w, Collection<Node> v) {
    a: if (v.contains(w))              1 + 2e (we accept 2e, 1 + e, e)
    b:     return;                     1 + 2e - n (we accept e - n)
    c: v.add(w);                       n
    d: for (Node node : w.neighbors)   n
    e:    viz(node, v);                2e
```

(c) Heaps (6 points) Array segment `hp[0..s-1]` is supposed to contain a heap: a tree with no holes that satisfies certain properties, as discussed in the lecture on priority queues and heaps. Write the body of the following function. (Recursion is not necessary and makes it harder.)

Hint: Our solution begins with this statement: `boolean maxHp= hp[0] > hp[1];`.

```
/** Return -1, 0, or 1 depending on whether hp[0..s-1] is a min-heap,
  * not a heap, or a max-heap.
  * Precondition: s > 1. The elements of hp[0..s-1] are all different.
  *                hp[0..s-1] represents a complete tree. */
static int isHeap(int[] hp, int s) {
```

```
boolean maxHp= hp[0] > hp[1];
for (int k= 1; k < s; k= k+1) {                 // Note. Many people tried to compare hp[k]
    if (maxHp != hp[(k-1)/2] > hp[k]) return 0;  // to its children instead of its parent.
}                                                // That's harder and much more work!
return maxHp ? 1 : -1;
```

```
}
```

7. Concurrency (10 Points)

(a) Deadlock (4 points).
The two threads to the right are run concurrently, sharing Lists `listG` and `listF`. Is it possible for these two threads to deadlock? Explain why or why not.

```
Thread 2: while (true) {
  synchronized(listF) {
    if (!listF.isEmpty()) listF.remove(0);
  }
  synchronized(listG) listG.add("May");
}

Thread 1: while (true) {
   synchronized(listG) {
      synchronized(listF) {
         while (!listG.isEmpty()) {
            listF.add(listG.remove(0));
} } } }
```

No, deadlock is not possible.
Thread 2 does not try to lock both lists at once, and neither thread blocks or spins except to acquire a lock

(b) Synchronization (6 points).
Consider the following class, an instance `enr` of which is shared among threads T1, T2, and T3:

```
class Enrollees {
  boolean signal;

  synchronized void what() throws InterruptedException {
    System.out.print("D ");
    Thread.sleep(2000);  // Sleep for 2 seconds
    System.out.print("E ");
    while (!signal) { wait(); }  // Wait for signal
    signal= false;
    System.out.print("F ");
} }
```

First, T1 calls `enr.what()`. Then, 3 seconds later, T2 calls `enr.what()`. 1 second later, T3 calls `enr.what()` and blocks. So far, the program has produced the output "D E D ". Answer the following questions about this point in time (each answer is some combination of "T1", "T2", and "T3", or the word "None"):

Which thread(s) are currently on the lockist? T3

Which thread(s) are currently on the waitlist? T1

Which thread(s) currently hold the lock for `enr`? T2