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.

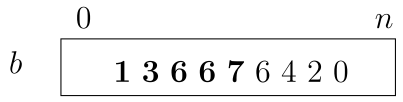
(signature)

1. Name (1 point)

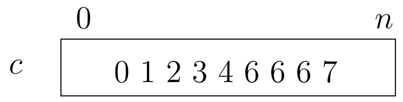
Is your name at the top of this page and your NetID at the top of pages 1..10 correct? If either is incorrect, or if you are missing pages, do not proceed; Contact the instructors immediately! Sign the Academic Integrity Statement before submitting your exam.

2. Loop Invariants (10 points)

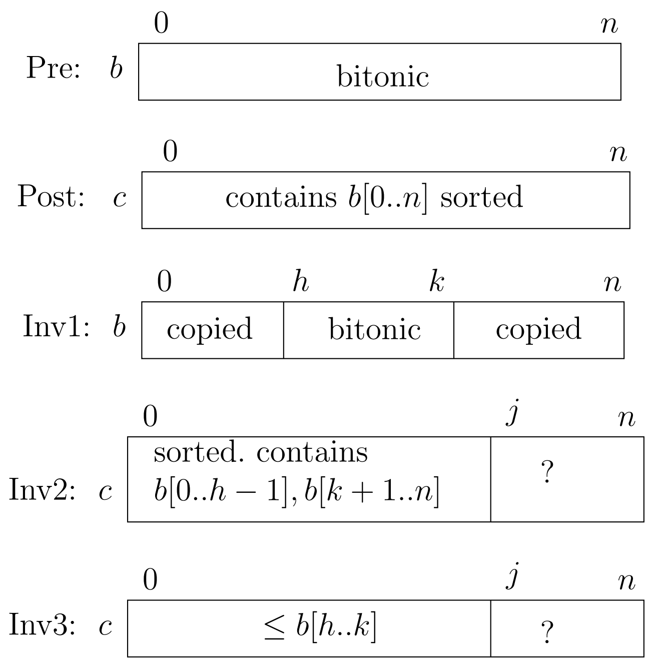
A sequence of integers is *bitonic* if it is first ascending and then descending. The following array is bitonic: The bolded values are ascending, the rest are descending.



We write a while-loop that copies bitonic segment $b[0..n]$ into a given array c in ascending order. Example: For the bitonic sequence $b[0..8]$ given above, $c[0..8]$ will be:



(Only part of array b is being sorted, not the whole array.) To the right are the precondition, postcondition, and invariant for this problem. The invariant has three parts.



(a) 3 points In the box to the right, write the initialization for the while-loop. Assume that variables have already been declared and that input parameters have been provided.

h= 0;
k= n;
j= 0;

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

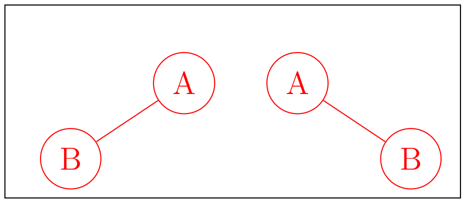
$h \leq k$ or $j \leq n$ or equivalent

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

```
if (b[h] < b[k])    // could be <=
      { c[j]= b[h]; h= h + 1; }
else { c[j]= b[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 printNums(List<Number> nums) {...}
```

Which of the argument types on the right can be passed to method `printNums()`? 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`.

A, C

`A. ArrayList<Number>`

`B. ArrayList<Integer>`

`C. LinkedList<Number>`

`D. List<Integer>`

`E. Number[]`

`F. Object`

(c) 4 points. Consider hashing with linear probing using a 7-element array `b[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 `b` after these values have been added: 11, 13, 20, 14. Do not be concerned with the load factor.

[20, 14, null, null, 11, null, 13]

(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. We want to sort an array of (x, y) pairs in lexicographic order, i.e. sort by the x dimension and break ties with the y dimension. For example, these pairs are in order: (1, 4), (1, 5), (2, 3).

We do this with two calls to existing procedure $DimSort(pairs, dimension)$, which sorts a pair-array in place by an indexed dimension (see right).

Between merge-sort and heap-sort, only one is appropriate to use for *DimSort*. Which one, and why?

```
// Sort by y dimension
DimSort(pairs, 1);

// sort by x dimension
DimSort(pairs, 0);
```

Merge-sort – must be stable

(f) 2 points. Consider a graph with n nodes and e edges. Algorithms A1 and A2 perform the same task but with different asymptotic complexity. Algorithm A1's time complexity is in $O(e\,n)$, while A2's is in $O(n^2 \lg(n))$. Which algorithm would you expect to run faster for *large, dense* graphs? Write its name (and nothing else) in the box.

A2

4. Object-Oriented Programming (14 points)

(a) 7 points You are working on a list of Profs (Professors) for Cornell. You are keeping track of what Profs taught together, and you want to sort Profs using that information. This will be done using class `Prof`. Two of its fields are shown to the right.
Example: prof `Muhl` taught 3 times with prof `Grie`, so the pair (object for `Muhl`, 3) occurs in field `taughtWith` in the object for `Grie`.

```
public class Prof
        implements Comparable<Prof> {
 /** The id of this prof */
 public int profId;

 /** A pair (i, val) is in taughtWith
   * if prof i taught with this prof
   * val times. */
 public HashMap<Prof, Integer> taughtWith;
```

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

```
/** Constructor: An instance with profId id who has taught with no one else */
public Prof(int id) {
```

```
profId= id;
taughtWith= new HashMap<Prof, Integer>();
```

```
 }

/** = number of times this prof has taught with every prof in taughtWith. */
  public int allTaught() {...}

/** Return -1 or 1 depending on whether this prof's allTaught()
    * is < or > pro's allTaught(). If this prof's allTaught() is
    * equal to pro's allTaught(), return -1, 0, or 1 depending on
    * whether this prof's profId is <, =, or > pro's profId.
 * Precondition: pro is not null */
public @Override int compareTo(Prof pro) {
```

```
int val= Integer.compare(allTaught(), pro.allTaught());
if (val == 0) return Integer.compare(profId, pro.profId);
return val;
```

```
}
```

(b) 7 points To the right is part of a class Heap, which implements a heap. Make it iterable by completing methods hasNext and next of inner class HeapIterator, below.

You don't need to know anything more about the implementation of a heap than is shown to the right; the heap elements are in array c[0..size-1].

Don't introduce any new fields, but you can use local variables. Don't worry about iterating through the items in order of priority.

```
class Heap<T> implements Iterable<T> {
   protected Item<T>[] c; // Heap is in
   protected int size;    // c[0..size-1]

   class Item<T> {
      protected T value;
      protected double priority;
      ...
   }

   public Iterator<T> iterator() {
      return new HeapIterator();
   }
```

```
/** An instance is an iterator over this heap. */
private class HeapIterator implements Iterator<T> {
    /** values  in c[0..k-1] have been enumerated. */
    private int k= 0;

    public HeapIterator() {} // Constructor: an iterator over this heap

    /** = "there is another value to enumerate." */
    public boolean hasNext() {
            return k < size;
    }

    /** Return the next value to enumerate, thus enumerating it.<br>
     * Throw a NoSuchElementException if there is no next element. */
    public T next() {
       if (!hasNext()) throw new NoSuchElementException();
       T val= c[k].value;    OR    k= k+1;
       k= k + 1;                   return c[k-1].value;
       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 A directed graph describes all flights in the U.S. Each node is an airport, and each edge weight gives the flight duration. Which algorithm is best suited to finding a flight from Portland to Ithaca with ≤ 1 stop (e.g. Portland→Ithaca or Portland→Dulles→Ithaca).

BFS

(c) 2 points Underground electricity conduits are being designed for a new subdivision. The nodes of an undirected graph are houses and the planned junction points. An edge exists between two nodes if it appears to be a viable conduit; it is labeled with the estimated cost of laying the line. What algorithm would you use minimize the cost of connecting all the nodes with conduit?

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.")

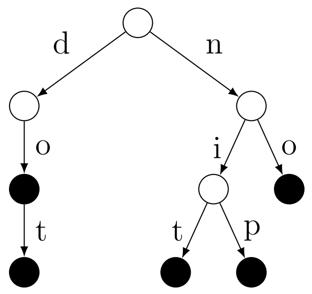
(d2) Each of the abstract algorithms (A1) and (A2) has a loop. Write down the number of iterations each loop takes (as a function of the number n of nodes and the number e of edges).

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 "do", "dot", "nit", "nip", and "no". 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 b[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, `b[0]` is for character 'a', `b[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 `b` corresponding to character `ch`:

b[$\boxed{\text{ch – 'a'}}$]

(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 an `ArrayList<Node>`. In method `vis` on the next page, each statement has a label on it, so we can talk about it.

Consider the call `vis(node, new HashSet<Node>())` of the method below. To the right of each statement in the body of `vis`, 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 u along unvisited paths
  * Precondition. v contains all visited nodes. */
static void vis(Node u, Collection<Node> v) {

    a: if (v.contains(u))                 1 + 2e (we accept 2e, 1 + e, e)

    b:     return;                        1 + 2e - n (we accept e - n)

    c: v.add(u);                          n

    d: for (Node node : u.neighbors)      n

    e:    vis(node, v);                   2e
```

(c) Heaps (6 points) Array segment $b[0..size-1]$ 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 maxHeap= b[0] > b[1];`.

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

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

}
```

7. Concurrency (10 Points)

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

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

Thread 2: while (true) {
   synchronized(listB) {
     if (!listB.isEmpty()) listB.remove(0);
   }
   synchronized(listA) listA.add("coin");
 }
```

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 `ros` of which is shared amoung threads T1, T2, and T3:

```
class Roster {
  boolean signal;

  synchronized void volunteer() throws InterruptedException {
    System.out.print("A ");
    Thread.sleep(2000);  // Sleep for 2 seconds
    System.out.print("B ");
    while (!signal) { wait(); }  // Wait for signal
    signal = false;
    System.out.print("C ");
} }
```

First, T1 calls `ros.volunteer()`. Then, 3 seconds later, T2 calls `ros.volunteer()`. 1 second later, T3 calls `ros.volunteer()` and blocks. So far, the program has produced the output "A B A ". 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 `ros`? T2