Name: NetID:

Prelim 2

CS 2110, 21 November 2019, 5:30 PM

	1	2	3	4	5	6	7	Total
Question	Name	Short Answer	Heaps	Trees	Collections	Graphs	Sorting	
Max	1	30	12	16	13	16	12	100
Score								
Grader								

The exam is closed book and closed notes. Do not begin until instructed.

You have **90 minutes**. Good luck!

Write your name and Cornell **NetID**, **legibly**, at the top of the first page, and your Cornell ID Number (7 digits) at the top of pages 2-8! There are 7 questions on 8 numbered pages, front and back. Check that you have all the pages. When you hand in your exam, make sure your pages are still stapled together. If not, please use our stapler to reattach all your pages!

We have scrap paper available. If you do a lot of crossing out and rewriting, you might want to write code on scrap paper first and then copy it to the exam so that we can make sense of what you handed in.

Write your answers in the space provided. Ambiguous answers will be considered incorrect. You should be able to fit your answers easily into the space provided.

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 make the problem more difficult. This does not mean that it's good style.

Academic Integrity Statement: I pledge that I have neither given nor received any unauthorized aid on this exam. I will not talk about the exam with anyone in this course who has not yet taken Prelim 2.

(signature)

1. Name (1 point)

Write your name and NetID, **legibly**, at the top of page 1. Write your Student ID Number (the 7 digits on your student ID) at the top of pages 2-8 (so each page has identification).

2. Short Answer (30 points)

(a) True / False (8 points) **Circle** T or F in the table below.

(a)	T	F	An algorithm with time complexity $O(n)$ will always run faster than one with time complexity $O(n^2)$.
(b)	T	F	In a dense graph, an algorithm whose running time is proportional to $\|E\|$ is preferable to one whose running time is proportional to $\|V\|$.
(c)	T	F	In order for a class to support for-each loops, it must implement `Iterator` and provide a method that returns an `Iterable`.
(d)	T	F	Although both quicksort and mergesort recursively sort sub-lists, quicksort's worst-case running time is $O(n^2)$, while mergesort's is $O(n \log n)$.
(e)	T	F	In a tree, every path between two nodes s and t in which all nodes are different is a shortest path from s to t.
(f)	T	F	Computing the outdegree of a node n is faster with adjacency lists than adjacency matrices.
(g)	T	F	Any class `K` can be used as a key in a `HashMap<K, V>`.
(h)	T	F	Swapping left and right subtrees in a BST breaks the BST property, but swapping them in a heap doesn't affect the heap-order property.

(b) Complexity (3 points) For each of the functions f below, state a function g such that f is $O(g)$ where $O(g)$ is as simple and tight as possible. For example, one could say that $f(n) = 2n^2$ is $O(n^3)$ or $O(2n^2)$, but the required answer is $O(n^2)$.

1. $f(n) = 5n^2 + 8n^3$
2. $f(n) = 2^n$ + n*log(n)
3. $f(n) = \frac{4n^3 - 3n^2 + 2}{5n}$

(c) GUI (3 points) What three steps are required to listen to an event in a Java GUI?

(d) Hashing (6 points) Consider a hash table of size 6, with elements numbered in 0..5. The hash function being used is:

$$H(k) = 2k.$$

Consider inserting these values into it, in order: 4, 0, 6, 5, 3, 7. (continued on next page.)

To the left, draw the table after inserting the values using open addressing with linear probing. To the right, draw the table after inserting the values (into an empty table) using chaining. Draw the linked lists as simply and as clearly as possible.

(e) Anonymous functions (6 points) State the property that interface `II` must satisfy in order to be used in an assignment statement like the one to the right:

```
II v=  an anonymous function;
```

Below, write an anonymous function that is equivalent to the function to the right:

```
int abs(int b) {
    return b < 0 ? -b : b;
}
```

(f) Method calls (4 points) Write the steps in executing the call m(5) on this procedure:

```
public static void m(int p) { ... }
```

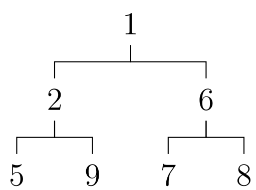
3. Heaps (12 Points)

(a) 2 points Suppose array $b[0..n-1]$ is a heap. To the right, write the index of the left child of node $b[i]$ (assuming it has one):

Answer

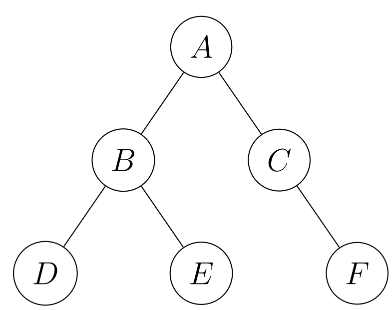
(b) 6 points To the right, draw the min-heap formed by inserting the following integers in order: $[21, 50, 7, 45, 10, 2, 12]$

(c) 4 points To the right, draw the min-heap that results from calling `poll()` on the following min-heap:



4. Trees (16 Points)

(a) 6 points Write the inorder and preorder traversals of the tree to the right (each on one line):



inorder:

preorder:

postorder:

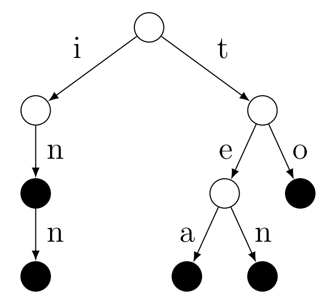
(b) 4 points To the right, construct a BST, adding nodes in the following order: $[4, 7, 2, 1, 3]$.

(c) 2 points What is the worst-case time complexity for the following, given a BST of n nodes?

Inserting a node:

Deleting a node:

(d) 4 points The *trie* data structure is a tree that maintains a set of words. Each edge is labeled with a lower-case letter, in `a..z`, so a node has at most 26 children.



Each node contains a boolean value to indicate whether the path from the root to that node represents a complete word. E.g. this allows both `"in"` and `"inn"` to be words, shown to the right. Here, the black nodes represent words in the set `{"in", "inn", "tea", "ten", "to"}`.

To look up a word like `"inn"`, begin at the root and follow the path of letters in the word. Assume that the child of a node corresponding to a letter can be referenced in constant time (the child is either a node or null, in case the child is empty).

Consider a trie with `n` strings of maximum length `m`. What is the tightest Big-O worst-case complexity of determining whether the trie contains a given word of length `k`? (Give your answer in terms of `n`, `m`, and/or `k`.)

Answer:

5. Collections (13 Points)

(a) 5 points Here are a few collection classes we have discussed:
`LinkedList, HashSet, ArrayList, Heap (without the extra HashMap used in A5)`
To the right of each of the following operations with the given data structure and size, give the tightest bound you can on worst-case time complexity. The answers are all one of: $O(1)$, $O(\log n)$, $O(n)$, $O(n \log n)$, $O(n^2)$, $O(n^2 \log n)$, and $O(n^3)$.

I. Merge sort on an ArrayList of size n^2:

II. Binary search on a (doubly) LinkedList of size n:

III. Search if there exists an element with given priority in a max-Heap of size n:

IV. Add an element to a HashSet (assume open addressing) with size n^3:

V. bubbleDown in a min-Heap of size n:

(b) 8 points Java has a class named `Deque`. It's a list that supports element insertion and removal at both ends —"deque" is short for "double ended queue". The start of the class declaration appears to the right, showing its fields. Assume the rest of the methods are there.

```
                                      /* TODO 5 */
public class Deque<E>                                      {
    /** Class invariant:
      * b[0..size-1] represents the deque.
      * b[0] is the first element and
      * b[size-1] is the last. */
    private E[] b;
    private int size;
    ...
```

Unfortunately, Java forgot to make this class `Iterable` and has asked us to do it. Below, we have stubbed in method `iterator` and fields and methods in class `MyIt`; these go in class `Deque`, of course. Complete (1) the body of method `iterator`; (2) the invariant for class `MyIt`, thus defining field `idx`; (3) the body of method `hasNext`; (4) the rest of the body of method `next`; —the latter two being written according to what you wrote for a class invariant. Finally, (5) Fix the first line of the declaration of class `Deque`, above to the right.

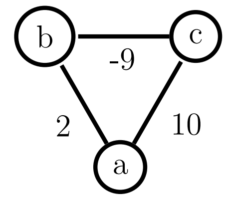
```
/** = an iterator over elements of this deque. */
public Iterator<E> iterator() {

    /*TODO 1:*/
}

/** An iterator over elements of this deque. */
private class MyIt implements Iterator<E> {
   /*TODO 2:*/
   //
   //
   private int idx= 0;

   /** = there is another value to be enumerated. */
   public @Override boolean hasNext() {
      /*TODO 3:*/

   }

/** Enumerate (return) the next value of this deque. <br>
 * Throw a NoSuchElementException if there is none. */
public @Override E next() {
   if (!hasNext()) throw new NoSuchElementException();
   /*TODO 4:*/

   }
}
```

6. Graphs (16 Points)

(a) 1 point One of BFS and DFS has a natural recursive implementation. Circle it:

DFS BFS

(b) 4 points For Dijkstra's shortest-path algorithm to work, all edge weights must be positive. To the right is a graph in which one edge weight is negative. Below are the settled set, frontier set, and d-values after the initialization and after the first iteration of the loop of the algorithm.



	Settled set	Frontier set	d-values
Initially:	{ }	{a}	d[a] = 0
After 1 iteration:	{a}	{b, c}	d[a] = 0, d[b] = 2, d[c] = 10

The algorithm was developed in terms of three invariants and a theorem proved from the invariant. State which of these four parts is not true of the settled set, frontier set, and d-values shown after one iteration. If you mention a part as being not true, you must state that part clearly.

(c) 4 points Both Kruskal and Prim's algorithms are additive, i.e. starting with no edges, edges are added one by one. Explain why an additive algorithm is better than a subtractive algorithm in terms of number of operations.

(d) 7 points Complete the following method. Make it recursive. Write "visit n" without explaining how to visit a node n and "n is visited" or "n is unvisited" to check whether n has been visited. You can also use an English phrase to loop through the neighbors of a given node.

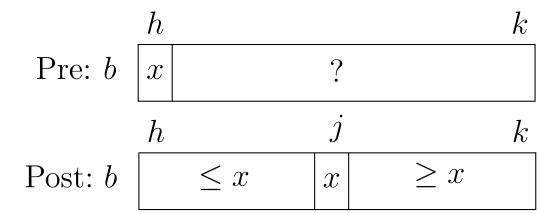
```
/** Return true if t is reachable along a completely unvisited path from s to t.*/
public boolean reachable(Node s, Node t) {
```

7. Sorting (12 Points)

(a) 4 points The following implementation of quicksort has two lines with errors. Cross out the two incorrect lines and rewrite them correctly. The specification of `partition` is shown in part (b).

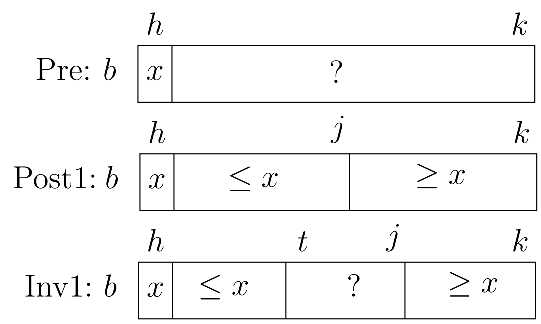
```
/** quicksort b[h..k]. */
public static void QS(int[] b, int h, int k) {
  if (k - h == 1) return;
  int j= partition(b, h, k);
  QS(b, h, j - 1);
  QS(b, j, k);
}
```

(b) 8 points Below is the header of method `partition`, using assertions `Pre` and `Post` on the right.




```
/** Given Pre, swap values of b[h..k] to
  * truthify Post and return j. */
static int partition(int[] b, int h, int k)
```

Method `partition` will use a loop with the same precondition but postcondition `Post1` and invariant `Inv1`, shown to the right. Then, one more statement truthifies `Post`. Below, write the method body in steps —all except the final return statement.



(b1) 1 point Write the loop initialization.

(b2) 1 point Write the loop condition (do not write "while").

(b3) 4 points Write the repetend. You can use a "Swap" statement.

(b4) 2 points Write a statement that, given `Post1` true, truthifies `Post`. "Swap" is allowed.