

Prelim 2

CS 2110, 23 April 2019, 5:30 PM

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | Total |
|----------|------|--------------|-------|-------|-------------|--------|---------|-------|
| Question | Name | Short Answer | Heaps | Trees | Collections | Graphs | Hashing | |
| Max | 1 | 13 | 12 | 20 | 19 | 30 | 5 | 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 (13 points)**(a) True / False (10 points)** Circle T or F in the table below.

| | | | |
|-----|---|---|--|
| (a) | T | F | A completely recursive Quicksort takes space $O(n)$ in the worst case, but this can be reduced to worst-case space $O(\log n)$. |
| (b) | T | F | A hash set implemented with chaining stores all values in a single linked list. |
| (c) | T | F | In a tree, leaves are nodes with 0 or 1 children. |
| (d) | T | F | The priorities in a heap must be doubles or ints. |
| (e) | T | F | It is possible to change the priority of any node in a heap (as implemented in A5) in expected time $O(\log n)$. |
| (f) | T | F | One advantage of using an Enum is that it is possible to use a foreach loop over the values in the Enum. |
| (g) | T | F | DFS starting from the root of a tree T and processing children left to right does a preorder traversal of T. |
| (h) | T | F | Adjacency lists take worst case $O(V)$ space. |
| (i) | T | F | A directed graph can be topologically sorted if and only if it has no cycles. |
| (j) | T | F | BFS is faster for dense graphs while DFS is faster for sparse graphs. |

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

3. Heaps (12 Points)

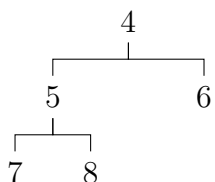
(a) **10 points** Complete method `kthLargest`, below. We suggest using class `Heap` from assignment A5, and we include the signatures of some relevant methods below.

```
/** Constructor: an empty heap with capacity 10.
 * It's a min-heap if isMin is true; otherwise, a max-heap. */
public Heap(boolean isMin);
/** Add v with priority p to the heap. */
public void add(E v, double p);
/** If this is a max-heap, return the heap value with highest priority.
 * If this is a min-heap, return the heap value with lowest priority. */
public E peek();
/** If this is a max-heap, remove and return the heap value with highest priority.
 * If this is a min-heap, remove and return heap value with lowest priority */
public E poll();

/** Precondition: 0 < k <= size of List s.
 * (Do not test the precondition.)
 * Return the kth largest value in s.
 * Note: If k = 1, return the largest; if k = 2, return the second largest, etc.
 * It should run in time  $O(n \log n)$  for s containing n values. */
public int kthLargest(int k, List<Integer> s) {
```

```
}
```

(b) **2 points** Draw the result after 3 is added to the min heap below:



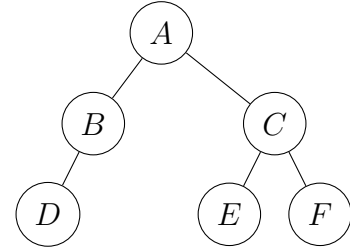
4. Trees (20 Points)

(a) 4 points

Write the inorder and postorder traversals of this tree:

inorder:

postorder:



(b) 4 points

Construct a BST, adding nodes in the following order: [5, 2, 0, 1, 4, 3]

(c) 2 points Suppose a BST has been constructed by inserting the odd integers in $1..n$ (n is odd) into an empty binary search tree, in order from least to greatest. What is the worst-case time complexity of inserting these two values:

- 6?
- $n + 1$?

(d) 10 points

A root-to-leaf path is a sequence of nodes starting with the root and ending with a leaf, with each non-leaf followed by one of its children.

Write method `maxPathSum`, below.

```
class Node {  
    public int val;  
    public List<Node> children;  
}
```

```
/** One can sum the values of a root-to-leaf path;  
 * return the largest such sum.  
 * Precondition: root is not null, and all node values are positive. */  
public int maxPathSum(Node root) {
```

```
}
```

5. Collections (19 Points)

(a) **5 points** Here are a few collections we have discussed:

LinkedList, HashSet, ArrayList, Heap (without the extra HashMap used in A5)

For each of the following operations, what would be the fastest data structure of the four to use (expected time)?

- I. Check if an element is in the collection.
- II. Access elements in the middle of the collection.
- III. Remove an arbitrary element from the collection.
- IV. Prepend an element.
- V. Process elements in a priority order.

(b) **10 points** Complete the body of method `getCounts`.

```
/** Return a map that records the number of occurrences of each char in b. */
public HashMap<Character, Integer> getCounts(char[] b) {

}

}
```

(c) **2 points** What is the expected complexity of this method if the size of `b` is `n`?

(d) **2 points** What is the worst-case complexity of this method if the size of `b` is `n`?

6. Graphs (30 Points)

(a) 6 points Consider an undirected graph. We want to color the nodes so that adjacent nodes do not have the same color, as usual. The available colors are C_1, C_2, C_3, \dots , in that order. The *Grundy number* of the graph is the maximum number of colors that can be used by a greedy coloring strategy, where “greedy” means *choosing the first available color*. We include two examples below.

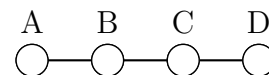
The graph to the right has a Grundy number of 3:

One order to get this number of colors is $A \rightarrow D \rightarrow B \rightarrow C$.

A and D get color C_1 , B gets color C_2 , and C gets color C_3 .

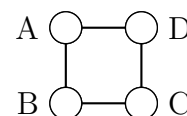
If we choose the order $A \rightarrow B \rightarrow C \rightarrow D$, only two colors are used:

A gets C_1 , B gets C_2 , C gets C_1 , and D gets C_2 .



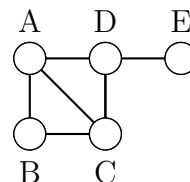
The graph to the right has a Grundy number of 2:

One order to get this number of colors is $B \rightarrow C \rightarrow D \rightarrow A$

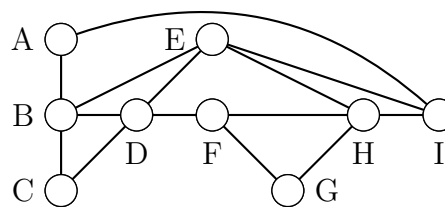
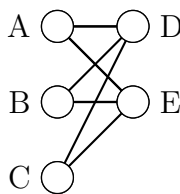
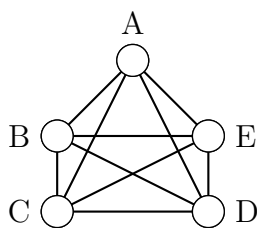


(a-i) What is the Grundy number of the graph to the right?

(a-ii) Give a selection order of vertices that leads to this many colors:



(b) 3 points For each of the three graphs below, state whether it is planar or not.



(c) 3 points State the theorem that is proved about the invariant in the development of Dijkstra's shortest path algorithm.

(d) 14 points Implement method `bfs`, given below. We provide parts of the specifications for `LinkedList` and `Node` that you may need.

```
/** A Linked List */
public class LinkedList<E> {
    public LinkedList<E>();           // Constructor: an empty list

    public boolean isEmpty();         // True if there are no items in the list

    public void addFirst(E item)      // Prepend item to the list

    public void addLast(E item);      // Append item to the list

    public E removeFirst(E item);     // Remove and return first item in the list

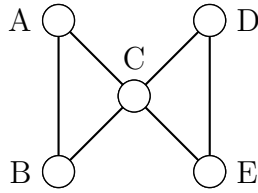
    public E removeLast(E item);     // Remove and return last item in the list
}

/** An instance is a node of this list. */
public class Node {
    public int id;                    // The node's id
    public List<Node> neighbors;      // list of neighbors of this node
    public boolean visited;           // true if this node has been visited
}

/** Return true if the graph starting from Node root contains a
 * node with id targetID. Use BFS to implement the search.
 * Precondition: root is not null. */
public boolean bfs(Node root, int targetID) {

}
```

(e) **4 points** Consider the following graph with the given edge weights.



| Edge | AB | AC | BC | CD | CE | DE |
|--------|----|----|----|----|----|----|
| Weight | 3 | 8 | 12 | 4 | 9 | 2 |

If we use Kruskal's algorithm to construct a minimum spanning tree, which edge will it choose first?

If we use Prim's algorithm to construct a minimum spanning tree (starting from vertex A), which edge will it choose first?

7. Hashing (5 Points)

(a) **3 points** Consider the hash table below, with fixed size 7, open addressing with linear probing, and hash function $x \rightarrow 3 * x$. Assume there is no resizing.

| | | | | | | |
|---|--|----|---|----|--|--|
| 7 | | 17 | 8 | 13 | | |
|---|--|----|---|----|--|--|

What does the hash table look like after the value 3 is added?

| | | | | | | |
|--|--|--|--|--|--|--|
| | | | | | | |
|--|--|--|--|--|--|--|

(b) **2 points** Consider a hash table with fixed size 5 that uses chaining. Using the hash function $x \rightarrow 2$, how many collisions will the sequence $\{6, 5, 4, 3, 2\}$ have? Assume there is no resizing.