Name: NetID:

# Prelim 2 Solution

**CS 2110, 23 April 2019, 7:30 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 hash set implemented with linear probing stores all values in an unordered bag. False, values must be stored in indexed buckets. |
| (b) | T | F | One advantage of using an Enum is that it is possible to use a foreach loop over the values in the Enum. True. |
| (c) | T | F | In a tree, leaves are nodes with 0 or 1 children. False. Leaves have 0 children only. |
| (d) | T | F | Priorities in a heap can be anything that is Comparable. True |
| (e) | T | F | It is possible to change the priority of any node in a heap (as implemented in A5) in expected time `O(1)`. False. |
| (f) | T | F | A completely recursive Quicksort takes space $O(n)$ in the worst case, but this can be reduced to worst-case space $O(\log n)$. True |
| (g) | T | F | BFS starting from the root of a tree T and processing children left to right does a level order traversal of T. True |
| (h) | T | F | Adjacency Matrices take worst case $O(\|V\|^2)$ space. True. |
| (i) | T | F | DFS is faster for dense graphs while BFS is faster for sparse graphs. False. |
| (j) | T | F | Any directed graph can be topologically sorted. False. The graph must be acyclic. |

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

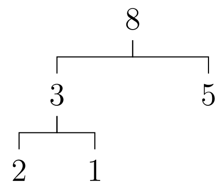
1. Have some class C implement an interface IN that is connected with the event.
2. In class C, override methods required by interface IN; these methods are generally called when the event happens.
3. Register an object of class C as a listener for the event. That object's methods will be called when event happens.

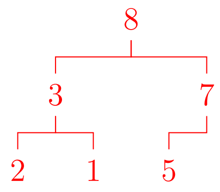
## 3. Heaps (12 Points)

**(a) 10 points** Complete method `kthSmallest`, 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 smallest value in s.
  * Note: If k = 1, return the smallest; if k = 2, return the second smallest, etc.
  * It should run in time O(n log n) for s containing n values. */
public int kthSmallest(int k, List<Double> s) {

  Heap<Double> heap= new Heap<Double>(true);
  for (Double do : s) {
      heap.add(do, do);
  }

 int ret= 0;                         OR   for (int i= 1; i < k; i++) {
 for (int i= 0; i < k; i++) {               heap.poll();
  ret= heap.poll();                       }
 }                                        return heap.peek();
 return ret;

}
```

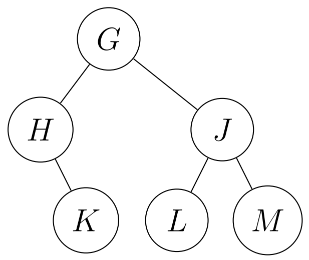
**(b) 2 points** Draw the result after 7 is added to the max heap below:





## 4. Trees (20 Points)

**(a) 4 points**
Write the inorder and postorder traversals of this tree:
Inorder: H K G L J M
Postorder: K H L M J G



**(b) 4 points** Construct a BST, adding nodes in the following order: [2, 6, 8, 3, 5, 9]

```
2
 \
  6
 / \
3   8
\    \
 5    9
```

**(c) 2 points** Suppose a BST has been constructed by inserting the even integers in 0..$n$ ($n$ is even) into an empty binary search tree, in order from least to greatest. What is the worst-case time complexity of inserting these two values:

- $n + 1$? Answer: O(n)
- 7? Answer: O(1). No matter how big n gets, need at most 4 tests.

**(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 `maxPathProduct`, below.

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

```
/** One can multiply together the values of a root-to-leaf path;
 *  return the largest such product.
 *  Precondition: root is not null, and all node values are >= 1. */
public int maxPathProduct(Node root) {

  int largestChild= 1;
  for (Node child : root.children) {
    largestChild= Math.max(largestChild, maxPathProduct(child));
  }
  return largestChild * root.val;

}
```

## 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. Process elements in a priority order. Heap

II. Access elements in the middle of the collection. ArrayList

III. Check if an element is in the collection. HashSet

IV. Remove an arbitrary element from the collection. HashSet

V. Append an element. LinkedList or ArrayList

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

```
/** Return a map that records the number of occurrences of each double in b. */
public HashMap<Double, Integer> getCts(double[] b) {
  HashMap<Double, Integer> cnts= new HashMap<Double, Integer>();
  for (double d : b) {
    Integer in= cnts.get(d);
    if (in != null) cnts.put(d, in + 1);
    else cnts.put(d, 1);
  }
  return cnts;
}
```

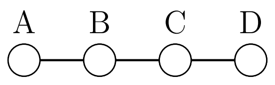
**(c) 2 points** What is the worst-case complexity of this method if the size of b is n?
O($n^2$).

**(d) 2 points** What is the best-case complexity of this method if the size of b is n?
O($n$).

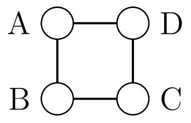
## 6. Graphs (30 Points)

**(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 C1, C2, C3, ..., 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.

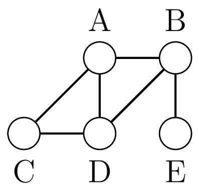
The graph to the right has a Grundy number of 3.
One order to get this number of colors is $A \rightarrow D \rightarrow B \rightarrow C$:
$A$ and $D$ get color C1, $B$ gets color C2, and $C$ gets color C3.
If we choose the order $A \rightarrow B \rightarrow C \rightarrow D$, only two colors are used:
$A$ gets C1, $B$ gets C2, $C$ gets C1, and $D$ gets C2.



The graph to the right has a Grundy number of 2:
One order to get this number of colors is $B \rightarrow C \rightarrow D \rightarrow A$

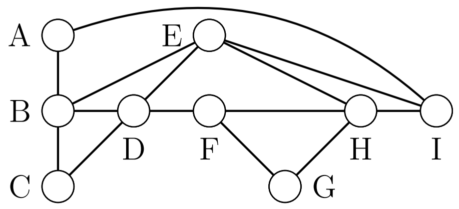


**(a-i)** What is the Grundy number of the graph to the right? 4.

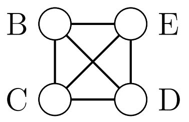


**(a-ii)** Give a selection order of vertices that leads to this many colors:
One sequence for achieving this is $E \rightarrow C \rightarrow A \rightarrow B \rightarrow D$.
Any sequence starting with $C \rightarrow A/D \rightarrow E$, $C \rightarrow E$, $E \rightarrow B \rightarrow C$, $E \rightarrow C$,
or the sequence $C \rightarrow A/D \rightarrow D/A \rightarrow E \rightarrow B$ will lead to a coloring number of 4.

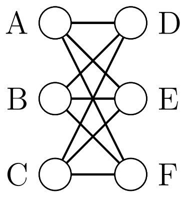
**(b) 3 points** For each of the three graphs below, state whether it is planar or not.



yes



yes



no

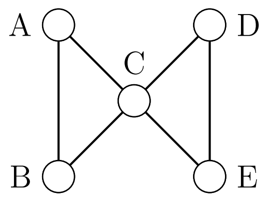
**(c) 3 points** Two of the three parts of the invariant for the development of Dijkstra's shortest path algorithm are written below. Write the missing part

1. All edges leaving the settled set end in the frontier set.
2. For a settled node s, at least one shortest path from v to s contains only settled nodes, and $d[s]$ is the distance of that shortest path from v to s.
3. For a node f in the frontier, at least one path from v to f contains only settled nodes, except for the last one, f, and $d[f]$ is the minimum distance of all such paths.

**(d) 14 points** Implement method `dfs`, given below, using iterative DFS. **Do not use recursion**. If you need a stack or queue, implement it using class LinkedList, partly specified below. We also provide parts of class Node.

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

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

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

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

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

  public E removeLast(T 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 DFS to implement the search.
  * Do not use recursion.
  * Precondition: root is not null. */
public boolean dfs(Node root, int targetID) {
  LinkedList<Node> stack= new LinkedList<>();
  stack.addLast(root);
  while (!stack.isEmpty()) {            // Inefficient in 2 ways,
      Node f= stack.removeLast();       // but simple. More efficient:
      if (f.id == targetID) return true;// 1. Don't put visited nodes
      if (!f.visited) {                 //    on stack.
          f.visited= true;              // 2. Test for root == targetId
          for (Node n : f.neighbors) {. //    in beginning and
             stack.addLast(n);          //    before putting f into stack.
          }
      }
  }
  return false;
}
```

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



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

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

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

## 7. Hashing (5 Points)

**(a) 3 points** Consider the hash table below, with fixed size 7, open addressing with linear probing, and hash function $x \to 2 * x$. Assume there is no resizing.

| | | | | | | |
|---|---|---|---|---|---|---|
| | 11 | 18 | 19 | | | 3 |

What does the hash table look like after the value 1 is added?

| | | | | | | |
|---|---|---|---|---|---|---|
| | 11 | 18 | 19 | 1 | | 3 |

**(b) 2 points** Consider a hash table with fixed size 3 that uses chaining. Using the hash function $x \to 6$, how many collisions will the sequence $\{0, 2, 4, 6\}$ have? Assume there is no resizing.
3 collisions.