

# Prelim 2

CS 2110, November 19, 2015, 5:30 PM

|          | 1             | 2               | 3          | 4                                  | 5     | 6      | Total |
|----------|---------------|-----------------|------------|------------------------------------|-------|--------|-------|
| Question | True<br>False | Short<br>Answer | Complexity | Searching<br>Sorting<br>Invariants | Trees | Graphs |       |
| Max      | 20            | 15              | 13         | 14                                 | 17    | 21     | 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** at the top of EACH page! There are 6 questions on 10 numbered pages, front and back. Check that you have all the pages. When you hand in your exam, make sure your booklet is 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 its good style.

## 1. True / False (20 points)

|    |   |   |  |
|----|---|---|--|
| a) | T | F | Calculating the depth of a tree always requires examining all of its nodes.  |
| b) | T | F | Let $t$ be a node of a tree. Determining whether $t$ is a leaf takes constant time.  |
| c) | T | F | Let $t$ be a BST with $n$ values, not necessarily balanced. The worst-case time for inserting a value into $t$ is $O(\log n)$ .  |
| d) | T | F | To obtain the values of a BST in ascending order, use a preorder traversal.  |
| e) | T | F | Given only the preorder and inorder traversals of a binary tree, one can construct the tree (uniquely).  |
| f) | T | F | The language of a grammar that contains no recursively defined nonterminals is finite.   |
| g) | T | F | A connected graph with $n$ nodes must have at least $n$ edges.   |
| h) | T | F | Topological sort works only if the graph it is processing is acyclic.  |
| i) | T | F | A bipartite graph must have an even number of nodes.   |
| j) | T | F | When using the adjacency matrix representation of a directed graph, to determine whether there exists an edge from node $u$ to node $v$ requires only constant time.   |
| k) | T | F | Breadth-first search can be written recursively or iteratively, but depth-first search can be written only iteratively.  |
| l) | T | F | Dijkstra's shortest-path on a graph with all edge weights being 2 is a depth-first search, and not a breadth-first search.   |
| m) | T | F | In the <b>best case</b> , it is not possible to use a comparison based sorting algorithm to sort an array in $O(n)$ time.  |
| n) | T | F | All JButtons in a GUI must be "listened to" with the same <code>actionPerformed</code> procedure; it's not possible to have different <code>actionPerformed</code> procedures for different buttons.                   |
| o) | T | F | Our code style guidelines say to declare all local variables used in a method at the beginning of the method, in order to save time allocating and deallocating space for them.  |
| p) | T | F | <code>Integer[]</code> is not a subclass of <code>Object[]</code> , just as <code>HashSet&lt;Integer&gt;</code> is not a subclass of <code>HashSet&lt;Object&gt;</code> .  |
| q) | T | F | Methods in a class <code>C</code> with an inner class <code>IC</code> can call methods in <code>IC</code> even though they are <code>private</code> .  |
| r) | T | F | If the value of function <code>equals(Object)</code> in a class depends only on fields <code>b</code> and <code>c</code> , function <code>hashCode()</code> in that class has to depend also on only those two fields. |
| s) | T | F | One can't use a "for-each" statement to process the elements of a <code>HashSet</code> because the elements of a <code>HashSet</code> are not ordered in any way.  |
| t) | T | F | The purpose of an object of class <code>Iterator</code> is to enumerate the values in some collection.   |



## 2.b Exception Handling 6 points

(i) **2 points** Consider the statement below, appearing in a method `m`, where `b` is an `int` array. Does its execution result in a `RuntimeException` being thrown out to the call of `m`? Write your answer and an explanation for it to the right of the statement.

```
try {
    int x= b[-5];
}
catch (Exception e) {
    throw new RuntimeException();
}
```

(ii) **4 points** To the right below, write down what is printed by the `println` statements during execution of the call `mm(1)`, where method `mm()` is defined as follows:

```
public static void mm(int x) {
    try {
        System.out.println("11");
        int b= 5/(x-1);
        System.out.println("12");
        return;
    } catch (RuntimeException e) {
        System.out.println("13");
        int c= 5/(x-1);
        System.out.println("14");
    }
    System.out.println("15");
    int d= 5/x;
    System.out.println("16");
    return;
}
```

### 3. Complexity (13 points)

(a) **4 points** For each of the functions  $f$  below, state the function  $g(n)$  such that  $f(n)$  is  $O(g(n))$ .  $g(n)$  should be as small as possible. (e.g. one could say that  $f(n) = 2n^2$  is  $O(n^3)$ , but the best answer, the one we want, is  $O(n^2)$ .)

(i)  $f(n) = \log(n) + n + n^3.$        $g(n) =$

(ii)  $f(n) = 2n + \frac{3000}{n} + 42.$        $g(n) =$

(iii)  $f(n) = 2^{n+12} + 400n^4.$        $g(n) =$

(iv)  $f(n) = 55$        $g(n) =$

(b) **3 points** State the tightest (smallest) asymptotic time complexity (in terms of  $n$ ) of the following statement sequence:

```
int s= 0;
for (int k= 0; k < n; k= k+1) {
    for (int j= k+5; j > k; j= j-1) {
        s= s + j*k;
    }
}
```

(c) **6 points** Give a formal proof that  $f(n) = 3000 + 2n$  is  $O(n)$ .

#### 4. Seaching, Sorting, and Invariants (14 points)

(a) 6 points Assume that this procedure has already been written:

```
/** Merge sorted segments b[h..j] and b[j+1..k] so that b[h..k] is sorted. */  
public static void merge(int[] b, int h, int j, int k) {...}
```

Below, write the body of procedure MS, completely in Java.

```
/** Sort b[h..k], using the mergesort algorithm. */  
public static void MS(int[] b, int h, int k) {
```

```
}
```

**(b) 8 points** Below is the precondition and postcondition of an algorithm to swap the negative values in  $b[h..k]$  into the beginning of  $b[h..k]$ . Note that  $h$  is not necessarily 0 and  $k$  is not necessarily  $b.length-1$ . Do not change  $h$  and  $k$ . You may assume the following procedure has already been written:

```
/** Swap b[i] and b[j]. */
public static void swap(int[] b, int i, int j) {...}
```

Complete procedure `p` below using a loop that follows the given loop invariant to accomplish this task.

|                |     |   |
|----------------|-----|---|
| Precondition:  | $b$ | $h$ <span style="float: right;"><math>k</math></span><br>unknown  |
| Postcondition: | $b$ | $h$ <span style="float: right;"><math>k</math></span><br><div style="display: flex; justify-content: space-around; width: 100%;"> <span><math>&lt; 0</math></span> <span><math>\geq 0</math></span> </div>  |
| Invariant:     | $b$ | $h$ <span style="margin-left: 100px;"><math>e</math></span> <span style="margin-left: 100px;"><math>f</math></span> <span style="float: right;"><math>k</math></span><br><div style="display: flex; justify-content: space-around; width: 100%;"> <span>unknown</span> <span><math>&lt; 0</math></span> <span><math>\geq 0</math></span> </div> |

```
/** Modify b[h..k] as defined above. */
public static void p(int[] b, int h, int k) {
```

```
}
```

## 5. Trees (17 points)

### 5.a Binary Search Trees (9 points)

Assume that each node of a binary search tree (BST), of class `Node`, has these fields:

- `Node left`: the left subtree (null if empty)
- `Node right`: the right subtree (null if empty)
- `int data`: the data of the node

Write the body of the following method:

```
/** Return true if v is in tree t and false otherwise.
 * Takes O(d) time, where d is the maximum depth of t
 * Precondition t is a BST --t being null means an empty tree */
public static boolean contains(Node t, int v) {

}
}
```

### 5.b Heaps (8 points)

Consider writing heapsort to sort an `int` array `b`. Complete the implementation of step (2) in the method below. You do not have to concern yourself with the implementation of step (1). In implementing step (2):

- Remember that `b[0]` is the root of the heap and is the largest value (it is a max-heap).
- Assume that function `int poll(int[] b, int k)` has already been written and can be used. It assumes that `b[0..k - 1]` is a heap, removes the root, does what is necessary to make `b[0..k - 2]` back into a heap, and returns the removed value.

```
/** Sort array b using heapsort */
public static void heapsort(int[] b) {
    // (1) Make b[0..b.length-1] into a max-heap (largest value ends up in b[0])
    heapify(b);

    // (2) Poll values from the heap and put them into their sorted position in b
    for (
        ) {

    }
}
}
```

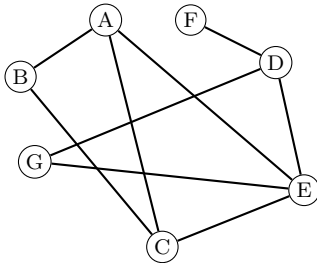


## 6. Graphs (21 points)

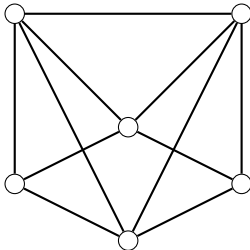
(a) **3 points** There are two basic ways to implement a graph: (1) an adjacency matrix and (2) an adjacency list. Let a graph have  $n$  nodes and  $e$  edges. Below, for each point, state to the right which representation of a graph has that property:

- (i) Uses space  $O(n + e)$ :
- (ii) Takes time  $O(n)$  to iterate over the edges that leave given node  $n$ :
- (iii) Takes time  $O(1)$  to determine whether there is an edge from node  $n_1$  to node  $n_2$ :

(b) **5 points** For the graph below, give a list of the nodes that are visited by the breadth-first search algorithm starting at node A, in the order visited. Whenever there is a choice of processing nodes in any way, process them in alphabetical order by their names. Example: to do something with nodes G, A, D, first do A, then D, and then G.



(c) **3 points** Is the following graph a planar graph? Write yes or no to its right.



Name:

NetID:

---

**(d) 3 points** State the difference between Kruskal's algorithm and Prim's algorithm for constructing a spanning tree of an undirected graph.

**(e) 7 points** Complete recursive algorithm `dfs`, given below. Do not be concerned about how marking occurs. You may simply say "mark `n`" and "if `n` is unmarked".

```
/** Mark all nodes that are reachable along unmarked paths from n.  
 * Precondition: n is unmarked. */  
public static void dfs(Node n) {
```

```
}
```