UNIVERSITY OF CALIFORNIA
Department of Electrical Engineering
and Computer Sciences
Computer Science Division

CS61B **P. N. Hilfinger**
Fall 2019

Final Examination (corrected) Solutions

READ THIS PAGE FIRST. *Please do not discuss this exam with people who haven't taken it.* Your exam should contain 9 problems on 17 pages. Officially, it is worth 46 points (out of a total of 200).

This is an open-book test. You have two hours and fifty minutes to complete it. You may consult any books, notes, or other non-responsive objects available to you. You may use any program text supplied in lectures, problem sets, or solutions. Please write your answers in the spaces provided in the test. Make sure to put your name, login, and TA in the space provided below. Put your login and initials *clearly* on each page of this test and on any additional sheets of paper you use for your answers.

Be warned: my tests are known to cause panic. Fortunately, this reputation is entirely unjustified. Just read all the questions carefully to begin with, and first try to answer those parts about which you feel most confident. Do not be alarmed if some of the answers are obvious. Should you feel an attack of anxiety coming on, feel free to jump up and run around the outside of the building once or twice.

Your name: _____ Login: _____

Your SID: _____ Discussion TA: _____

Login of person to your Left: _____ Right: _____

Please sign:

I pledge my honor that during this examination, I have neither given nor received assistance.

Signature: _____

Reference Material. The excerpts below are not intended to be comprehensive. Feel free to use other methods as well.

```
Methods of System:
   void arraycopy(src, k0, dest, k1, len): Copy len elements from src,
      starting at index k0, to dest, starting at index k1.

Methods of implementations of Collection<T>, Set<T>, List<T>:
   int size(): Number of entries.
   boolean isEmpty(): True iff .size() is zero.
   boolean contains(v): True iff v is an element.
   Iterator<T> iterator(): Iterator over all elements.
   boolean remove(v): Remove first element that equals v, returns true
                      iff it was present.
   void clear(): Remove all elements.

Methods of implementations of List<T>, Set<T>:
   boolean add(v): Add element v to collection. Adds to end for lists.
           Returns true iff collection is changed (may not be for Sets).

Methods of implementations of List<T>:
   T get(k): Return element indexed by k.
   boolean add(k, v): Add element v at index k.
   T remove(k): Remove and return element k (k an int).

Methods of implementations of Map<Key, Value>
   T get(k): Get element mapped by k, or null.
   boolean containsKey(k): True iff k is mapped.
   void put(k, v): Set element mapped by k to v.
   Set<Key> keySet(): A set of all keys in map.
   Collection<Value> values(): A collection of all mapped-to values.

Methods of functional interface java.util.function.Consumer:
   void accept(v): Execute the method overriding accept.
```

1. [3 points] Fill in the blanks in the following to fulfill the comments.

```
import java.util.regex.Pattern;
import java.util.regex.Matcher;

class Args {
    /** A Java pattern string that matches a comma-separated list of words
     *  (where a word is a sequence of one or more letters in the
     *  English alphabet of any case), and that captures the first word
     *  that contains at least one capital letter (not necessarily at the
     *  beginning).  There must be at least one such word. For
     *  example, this would match
     *       "joe,bob,laTex,zorch"
     *  and capture "laTex" in one of its groups, but would not match any of
     *       ",cat,Dog,bear"    "ax,hammer,saw"   "left,,right,Middle"
     */
    static final Pattern PATN =

      Pattern.compile("([a-z]+,)*([a-z]*[A-Z][a-zA-Z]*)(,[a-zA-Z]*)           ");

    /** Prints the first word in TEXT, a comma-separated list of
     *  words, that contains an upper-case letter. */
    public static void main(String text) {
        Matcher matcher = PATN.matcher(text);
        matcher.matches();

        System.out.println(matcher.group(2          ));
    }

}
```

2. [3 points] These questions concern the bit operators

```
&  |  ^  ~  <<  >>  >>>
```

Assume in each case that `x>0` ***is a Java*** `int` ***variable.***

a. What can you say about `x` if

```
((x - 1) & x) == 0 ?
```

In your answer, make no reference to bits or bit operations.

x is a power of 2.

b. What can you say about `x` if

```
((x>>4)<<4) == x ?
```

In your answer, make no reference to bits or bit operations.

x is evenly divisible by 16.

c. Fill in the blank to make the equation true, using **only** integer literals, bit operators, the `==` operator, the `!=` operator, the variable `x`, and parentheses:

```
(x > 31) = x >> 5 != 0
```

3. [8 points] Fill in the appropriate bubbles or provide short answers to the following.

a. Consider a max-heap stored in an array (as in class) containing N elements. What is the minimum number of elements that need to be examined to be sure of always finding the smallest?

◯ 1 ◯ $\approx \lg N$ ◯ $\approx \sqrt{N}$ ● $\approx N/2$ ◯ $\approx N$

b. Suppose that instead of each node in a max-heap having at most two children, it had at most four, while obeying the same heap property. What happens to the maximum time (not asymptotic time) required to insert a new value into this heap?

● Decreases by factor of 2 because of the tree's height.
◯ Increases because more comparisons are needed at each node.
◯ Stays the same.
◯ Depends on how expensive comparisons are.

c. Under the same assumptions as in part (b), what happens to the worst-case time for removing the largest key and re-establishing the heap property (as compared to the binary heaps we considered in class)?

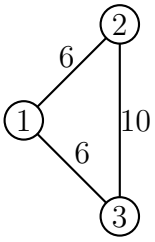
◯ Decreases by factor of 2 because of the tree's height.
◯ Increases because more comparisons are needed at each node.
● Stays the same.
◯ Depends on how expensive comparisons are.

d. Consider a dictionary of randomly selected lower-case words stored in a hash table. For a word W, the hash function in this table is W`.charAt(0)`. Assuming values of this hash function are evenly distributed for this set of words, what is the average look-up time for an item in the table over this same set of words?

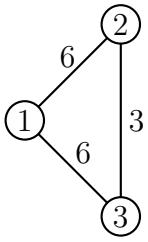
◯ $\Theta(1)$ ◯ $\Theta(\lg N)$ ◯ $\Theta(\sqrt{N})$ ● $\Theta(N)$

e. Show a labeled graph with edge weights with three vertices (labeled 1, 2, 3) in which the edges in the shortest-path tree from node 1 (as produced by Dijkstra's algorithm) are the same as those in the minimum spanning tree. Show a graph where the two are different.

Same:



Different:



f. In lecture, we saw how to use a post-order depth-first search on a directed graph with reversed edges to find a topological sort of the graph. Where in this algorithm could you add a check to quickly discover that no topological sort exists because the original graph was cyclic, and how would you do this check?

In the course of the traversal, I finish with a node (it has no unmarked successors), but at least one of its marked successors has not been added to the sequence of output nodes.

g. Which sorting algorithm most closely corresponds to the trie data structure?

◯ Insertion sort ◯ Selection sort ◯ Merge sort ◯ LSD radix sort
● MSD radix sort ◯ Quicksort

h. What data structure could you use to implement a double-ended priority queue—that is, an object supporting the operations of adding an element, removing the smallest element, and removing the largest element, with each taking $O(\lg N)$ time in the worst case for a queue with N elements?

A red-black tree.

4. [6 points] In the following questions, notations such as $A \subseteq B$ mean that every function in the set of functions A is also in the set of functions B. Likewise, $A = B$ means that A and B are the same set of functions. Fill in the appropriate bubbles.

a. True or false: $\Theta(2^N) \subseteq \Theta(3^N)$.

◯ True ● False

b. True or false: $\Theta(2^N) = \Theta(2^{N+3})$

● True ◯ False

c. True or false: $\Omega(\lg N) \subseteq \Omega(1000 \cdot N)$

◯ True ● False

d. What is the worst-case running time of the call `f(N, 1)` as a function of `N`?

```
int f(int M, int k) {
    if (k >= 1 && M < 1) {
       return 1;
    } else if (k >= 1) {
       return f(M - 1, k) + f(M, k - 1);
    } else if (M < 1) {
        return 1;
    } else {
        return M * f(M - 1, k);
    }
}
```

◯ $\Theta(1)$ ◯ $\Theta(\lg N)$ ◯ $\Theta(N)$ ◯ $\Theta(N \lg N)$ ● $\Theta(N^2)$ ◯ $\Theta(N^3)$ ◯ $\Theta(2^N)$

e. What is the worst-case running time of the call `f(N)` as a function of N? Assume that the function `h` takes constant time.

```
int f(int M) {
    for (int j = 0; j < M; j += 1) {
        if (h(j, M)) {
            for ( ; M >= j; M -= 1) {
                System.println(j);
            }
        } else {
            System.println(j);
        }
    }
}
```

◯ $\Theta(1)$ ◯ $\Theta(\lg N)$ ● $\Theta(N)$ ◯ $\Theta(N \lg N)$ ◯ $\Theta(N^2)$ ◯ $\Theta(N^3)$ ◯ $\Theta(2^N)$

f. What is the worst-case running time of the call `f(A)` as a function of N, the length of `A`? Assume calls to `h` and `p` take constant time.

```
void f(int[] B) {
    g(B, 0, B.length);
}

void g(int[] C, int L, int U) {
    if (L < U) {
        int M = (L + U) / 2;
        if (p(C, L, U)) {
            g(C, M + 1, U);
        } else if (p(C, L + 1, U - 1)) {
            g(C, L, M - 1);
        }
        for (int j = L; j < U; j += 1) {
            h(C, j, L, U);
        }
    }
}
```

◯ $\Theta(1)$ ◯ $\Theta(\lg N)$ ● $\Theta(N)$ ◯ $\Theta(N \lg N)$ ◯ $\Theta(N^2)$ ◯ $\Theta(N^3)$ ◯ $\Theta(2^N)$

5. [1 point] One of the following mountains does not belong. Which?

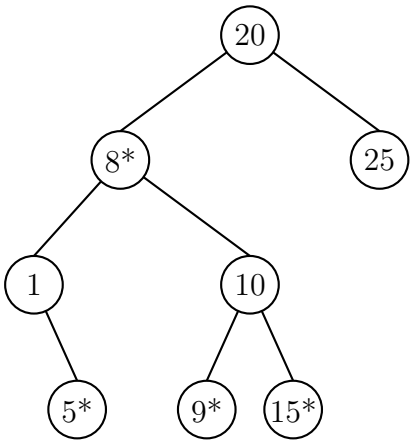
Denali, Haleakala, Caloris Montes, Euboea Montes, Maat Mons, Olympus Mons, Piccard Mons, Wright Mons

Euboea Montes is on a moon (Io), whereas all the others are on planets or dwarf planets.

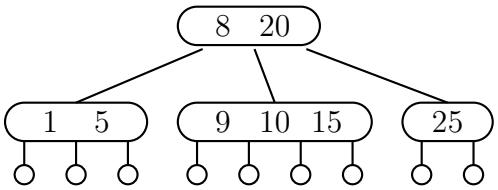
6. [6 points]

a. Show the unique (2, 4) tree corresponding to the red-black tree below. Red nodes are indicated by an asterisk ('∗') after the node label.

Red Black Tree



(2,4) Tree



b. What is the worst-case (maximum) height of a (2, 4) tree containing 15 keys? (Don't count the empty leaf nodes at the bottom: a tree all of whose keys are in the root thus has height 0).

● 3 ○ 4 ○ 5 ○ 6 ○ 14 ○ 15 ○ 16

c. True or false: it is possible to create a (2, 4) tree of maximum height (as defined above) for any $N > 0$ by choosing some sequence of N distinct keys to insert, starting from an empty tree, without any intervening removals.

○ True ● False

7. [6 points] Below you will find some intermediate steps in performing various sorting algorithms on the same input list. The steps do not necessarily represent consecutive steps in the algorithm (that is, many steps are missing), but they are in the correct sequence. For each of them, select the algorithm (by filling in the appropriate bubble) from among the following choices: insertion sort, straight selection sort, mergesort, quicksort, heapsort, LSD radix and MSD radix sort. For quicksort, the pivot is the median value of the first, last, and middle element (specifically, the index of the middle element is the average of the indices of the left and right elements, rounded down).

In all these cases, the final step of the algorithm will be the sorted sequence:

```
499, 1428, 1803, 2231, 2402, 3243, 3595, 4226, 4728, 5827, 6269, 6566, 7283, 9670
```

but it might not be shown.

Input List:

```
9670, 7283, 6566, 6269, 5827, 4728, 4226, 3595, 3243, 2402, 2231, 1803, 1428,  499
```

a. ◯ Insert. ◯ Select ● Merge ◯ Quick ◯ Heap ◯ LSD ◯ MSD

```
9670, 7283, 6566, 6269, 5827, 4728, 4226, 3595, 3243, 2402, 2231, 1803, 1428,  499
7283, 9670, 6269, 6566, 5827, 4728, 4226, 3595, 3243, 2402, 2231, 1803, 1428,  499
6269, 6566, 7283, 9670, 5827, 4728, 4226, 3595, 3243, 2402, 2231, 1803, 1428,  499
6269, 6566, 7283, 9670, 4226, 4728, 5827, 3595, 3243, 2402, 2231, 1803, 1428,  499
4226, 4728, 5827, 6269, 6566, 7283, 9670, 3595, 3243, 2402, 2231, 1803, 1428,  499
4226, 4728, 5827, 6269, 6566, 7283, 9670,  499, 1428, 1803, 2231, 2402, 3243, 3595
```

b. ◯ Insert. ◯ Select ◯ Merge ◯ Quick ● Heap ◯ LSD ◯ MSD

```
9670, 7283, 6566, 6269, 5827, 4728, 4226, 3595, 3243, 2402, 2231, 1803, 1428,  499
9670, 7283, 6566, 6269, 5827, 4728, 4226, 3595, 3243, 2402, 2231, 1803, 1428,  499
 499, 7283, 6566, 6269, 5827, 4728, 4226, 3595, 3243, 2402, 2231, 1803, 1428, 9670
7283, 6269, 6566, 3595, 5827, 4728, 4226,  499, 3243, 2402, 2231, 1803, 1428, 9670
1428, 6269, 6566, 3595, 5827, 4728, 4226,  499, 3243, 2402, 2231, 1803, 7283, 9670
6566, 6269, 4728, 3595, 5827, 1803, 4226,  499, 3243, 2402, 2231, 1428, 7283, 9670
1428, 6269, 4728, 3595, 5827, 1803, 4226,  499, 3243, 2402, 2231, 6566, 7283, 9670
 499, 1428, 1803, 2231, 2402, 3243, 3595, 4226, 4728, 5827, 6269, 6566, 7283, 9670
```

c. ● Insert. ◯ Select ◯ Merge ◯ Quick ◯ Heap ◯ LSD ◯ MSD

```
9670, 7283, 6566, 6269, 5827, 4728, 4226, 3595, 3243, 2402, 2231, 1803, 1428,  499
7283, 9670, 6566, 6269, 5827, 4728, 4226, 3595, 3243, 2402, 2231, 1803, 1428,  499
6269, 6566, 7283, 9670, 5827, 4728, 4226, 3595, 3243, 2402, 2231, 1803, 1428,  499
2402, 3243, 3595, 4226, 4728, 5827, 6269, 6566, 7283, 9670, 2231, 1803, 1428,  499
```

Continued on next page.

d. ◯ Insert. ● Select ◯ Merge ◯ Quick ◯ Heap ◯ LSD ◯ MSD

```
9670, 7283, 6566, 6269, 5827, 4728, 4226, 3595, 3243, 2402, 2231, 1803, 1428,  499
 499, 7283, 6566, 6269, 5827, 4728, 4226, 3595, 3243, 2402, 2231, 1803, 1428, 9670
 499, 1428, 6566, 6269, 5827, 4728, 4226, 3595, 3243, 2402, 2231, 1803, 7283, 9670
 499, 1428, 1803, 6269, 5827, 4728, 4226, 3595, 3243, 2402, 2231, 6566, 7283, 9670
 499, 1428, 1803, 2231, 5827, 4728, 4226, 3595, 3243, 2402, 6269, 6566, 7283, 9670
 499, 1428, 1803, 2231, 2402, 4728, 4226, 3595, 3243, 5827, 6269, 6566, 7283, 9670
 499, 1428, 1803, 2231, 2402, 3243, 4226, 3595, 4728, 5827, 6269, 6566, 7283, 9670
 499, 1428, 1803, 2231, 2402, 3243, 3595, 4226, 4728, 5827, 6269, 6566, 7283, 9670
 499, 1428, 1803, 2231, 2402, 3243, 3595, 4226, 4728, 5827, 6269, 6566, 7283, 9670
 499, 1428, 1803, 2231, 2402, 3243, 3595, 4226, 4728, 5827, 6269, 6566, 7283, 9670
 499, 1428, 1803, 2231, 2402, 3243, 3595, 4226, 4728, 5827, 6269, 6566, 7283, 9670
 499, 1428, 1803, 2231, 2402, 3243, 3595, 4226, 4728, 5827, 6269, 6566, 7283, 9670
```

e. ◯ Insert. ◯ Select ◯ Merge ◯ Quick ◯ Heap ● LSD ◯ MSD

```
9670, 7283, 6566, 6269, 5827, 4728, 4226, 3595, 3243, 2402, 2231, 1803, 1428,  499
9670, 2231, 2402, 7283, 3243, 1803, 3595, 6566, 4226, 5827, 4728, 1428, 6269,  499
2402, 1803, 4226, 5827, 4728, 1428, 2231, 3243, 6566, 6269, 9670, 7283, 3595,  499
4226, 2231, 3243, 6269, 7283, 2402, 1428,  499, 6566, 3595, 9670, 4728, 1803, 5827
```

f. ◯ Insert. ◯ Select ◯ Merge ● Quick ◯ Heap ◯ LSD ◯ MSD

```
9670, 7283, 6566, 6269, 5827, 4728, 4226, 3595, 3243, 2402, 2231, 1803, 1428,  499
 499, 3595, 3243, 2402, 2231, 1803, 1428, 4226, 6566, 6269, 5827, 4728, 9670, 7283
 499, 3595, 3243, 2402, 2231, 1803, 1428, 4226, 4728, 6269, 5827, 6566, 9670, 7283
 499, 3595, 3243, 2402, 2231, 1803, 1428, 4226, 4728, 6269, 5827, 6566, 7283, 9670
 499, 3595, 3243, 2402, 2231, 1803, 1428, 4226, 4728, 5827, 6269, 6566, 7283, 9670
 499, 1428, 1803, 2231, 2402, 3243, 3595, 4226, 4728, 5827, 6269, 6566, 7283, 9670
```

8. [4 points]

a. A student's project 3 failed on the command

`java gitlet.Main init`

We found the following in `Main.java`:

```
public static void main(String... args) throws IOException {
    if (args.length == 0) {
        Utils.error("Must have at least one argument");
    }
    String command = args[0];
    if (command == "init") {
        init();
    }
    if (command == "add") {
        String file = args[1];
        add(file);
    }
    if (command == "commit") {
        ...
```

What might have caused the failure the student saw?

Use of == for string comparison in Java.

b. In this same code, why is the first line problematic (i.e., `"static main..."`)?

Programs should not allow deliberately unhandled exceptions to be seen by clients; exceptions are for internal signalling.

c. A student writes an instance method containing the lines:

```
for (Edge e : _edges) {
    pruneGraph(e);
}
```

Somewhere during its execution, a `ConcurrentModificationException` occurs. What could be wrong and how might it be fixed?

Possibly, `pruneGraph` is modifying `_edges`, which interferes with the iteration over that object (possibly an `ArrayList`). One can fix this in various ways; a simple brute-force measure is to replace the **for**-loop header with

```
for (Edge e :  new ArrayList<Edge>(_edges))
```

d. A student submitted a bug report for Project 2 (Tablut) that said his AI seemed to make illegal moves, even though he had extensively tested his `legalMoves` and `isLegal` methods. His `Board` class contained the following:

```
...
class Board {
    ...
    /** Copies MODEL into me. */
    void copy(Board model) {
        if (model == this) {
            return;
        }
        ...
        _winner = model.winner();
        _turn = model.turn();
        _moveCount = model._moveCount;
        _moveCountLimit = model._moveCountLimit;
        _repeated = model._repeated;
        _undoStack = model._undoStack;
        _board = new Piece[SIZE][SIZE];
        System.arraycopy(model._board, 0, _board, 0, _board.length);
    }
    ...
}
```

What problems are evident in this code snippet? To find them all you may have to infer what the instance variables are used for. Assume that their names correspond to their usages.

First `_undoStack` is likely supposed to be some sort of list or array object containing information for undoing moves. As such, copying it from `model` by simple assignment will lead to two copies sharing the same object, which could cause trouble. Likewise, although `_board` is created using new objects (distinct from those of `model`), the effect of the `arraycopy` is to replace all its rows with pointers to those of `model`, again sharing structure.

9. [10 points] Consider the following general-purpose interface defining accessors to a directed graph:

```
/** Represents a directed graph whose vertices are labeled with
 *  positive integers. */
public interface Graph {
    /** Return an iterator over the vertex labels of this Graph. */
    Iterator<Integer> vertices();

    /** Return an iterator over values v1 such that (V, v1) is an edge
     *  in this graph. */
    Iterator<Integer> successors(int v);
}
```

a. The class `SimpGraph` is intended to be an implementation of `Graph` whose vertex labels are arbitrary positive integers and whose vertices and edges are supplied by `add` methods. Complete the implementation on the next page (don't forget that it must implement `Graph`). It is not necessary to do error checking. [Hint: with judicious use of the Java library, each method's and constructor's implementation contains a single statement, although you may use more if you desire. The reference material at the beginning of the exam summarizes some useful classes.]

b. The `Traverser` class takes a `Graph` in its constructor and supplies a method that traverses all vertices reachable by a directed path from a given staring vertex, applying an arbitrary function, supplied as the `accept` method of a functional object. For example,

```
new Traverser(G).traverseReachable(1, x -> System.out.println(x));
```

should print all vertex labels in G that can be reached from vertex 1. Here, Java converts the lambda expression (`x ->...`) into a `Consumer` object whose `accept` method prints its argument.

Fill in the implementation of `Traverser` on page 17. You may assume that you can freely use instance variables to carry information during an execution of `traverseReachable`. You need not use all blank lines.

```
import java.util.ArrayList;
import java.util.HashMap;
import java.util.Iterator;

public class SimpGraph implements Graph {

    /** A graph, initially empty. */
    public SimpGraph() {
    }

    @Override
    public Iterator<Integer> vertices() {
        return _edges.keySet().iterator();
    }

    @Override
    public Iterator<Integer> successors(int v) {
        return _edges.get(v).iterator();
    }

    /** Add a vertex with label V > 0.  V must not already be a vertex. *
    public void add(int v) {
        _edges.put(v, new ArrayList<Integer>());
    }

    /** Add a directed edge (V0, V1), where V0 and V1 are vertices in thi
     *  graph. The edge must not currently exist.  */
    public void add(int v0, int v1) {
        _edges.get(v0).add(v1);
    }

    /** A mapping from vertices to successors. */
    private HashMap<Integer, ArrayList<Integer>> _edges = new HashMap<>()

}
```

```
import java.util.function.Consumer;
import java.util.Iterator;

public class Traverser {

    public Traverser(Graph G) {
        _G = G;
    }

    /** Apply FUNC.accept to each vertex label in my graph exactly
     *  once for each label that is reachable from V0. */
    public void traverseReachable(int v0, Consumer<Integer> func) {
        _func = func;
        _marked.clear();
        traverse(v0);
    }

    public void traverse(int start) {
        if (_marked.add(start)) {
            _func.accept(start);
            for (Iterator<Integer> i = _G.successors(start); i.hasNext();
                traverse(i.next());
            }
        }
    }

    private Graph _G;
    private HashSet<Integer> _marked = new HashSet<>();
    private Consumer<Integer> _func;
}
```