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

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

Test #2

READ THIS PAGE FIRST. *Please do not discuss this exam with people who haven't taken it.* Your exam should contain 8 problems on 11 pages. Officially, it is worth 17 points (out of a total of 200).

This is an open-book test. You have 110 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: ____________________

Login of person to your Left: ______________ Right: ______________

Discussion TA: ________________________________

1. [2 points] Consider the following class, which uses our Project #1.

```
import enigma.Alphabet;
import enigma.Machine;

class Dicey {
    private Machine _mach;
    private int _data;
    private static Alphabet digits = new Alphabet("0123456789-");

    Dicey(int data) {
        _mach = new enigma.Machine(digits, ...); // A
        _mach.insertRotors(some rotor list);       // B
        _mach.setRotors("0000");                   // C
        _data = data;                              // D
    }

    @Override
    public int hashCode() {
        String digitString = Integer.toString(_data);         // E
        return Integer.parseInt(_mach.convert(digitString)); // F
    }


    @Override
    public boolean equals(Object other) {
         return _data == ((Dicey) other)._data;   // G
    }
}
```

(a) When used with a `HashSet`, the `Dicey` class does not work properly. Why not?

Its result changes each time a `Dicey` object is presented to the `HashSet`.

(b) How can we fix the problem in (a)? There may be multiple answers for a particular line; choose any one.

Move line A ☐ B ☐ [✓ C] D ☐ E ☐ F ☐ G ☐
to [✓ before] after ☐
line A ☐ B ☐ C ☐ D ☐ [✓ E] [✓ F] G ☐.

(c) Suppose we add a new method to the (corrected) class from (b):

```
void set(int s) {
    _data = s;
}
```

This causes problems with uses of `Dicey` objects in a `HashSet`. Why?

When one calls the method on a `Dicey` that is in a `HashSet`, its hashed value changes, so that the `HashSet` will no longer look for it in the right bucket.

(d) Suppose we replace lines E and F in the corrected class from (b) with

```
return _data;
```

Will this work in a `HashSet`? If not, why not?

Yes, it will work.

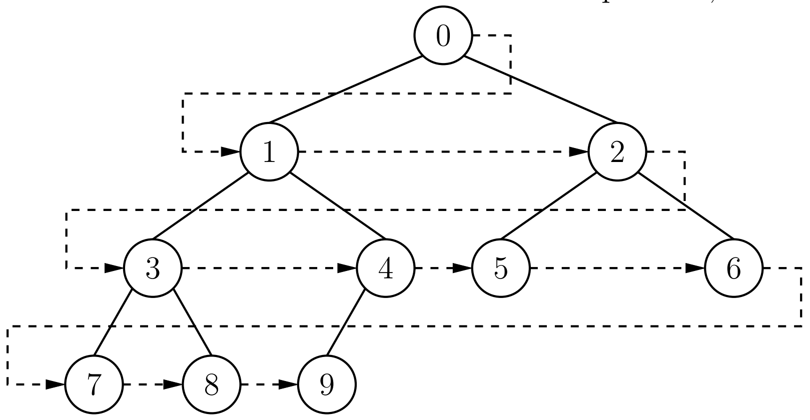
(e) Suppose we remove the overriding of `.equals` in the corrected class from (b). Will the resulting class work in a `HashSet`? If not, why not?

Yes, it will work.

(f) Suppose that we remove the overriding of `hashCode`. in the corrected class from (b). Will the resulting class work in a `HashSet`? If not, why not?

No; different `Dicey` objects that compare equal will hash to different buckets.

2. [3 points] In a *complete binary tree,* every level is filled, except possibly the last, and the last level has all nodes as far to the left as possible, like this:



The dashed arrows here link the nodes in level order. Given a complete binary tree such as this, your job is to supply links representing the dashed arrows. We have a class called `BinTree` that is similar to a normal binary tree with left and right pointers, except that it also has an extra pointer called "next" that is supposed to point to the next `BinTree` node in the tree in level order. Fill in the `linkInLevelOrder` method on the next page to fulfill its comment.

```java
class BinTree {
    public BinTree left, right, next;
    // Possibly other stuff

    /** Link the tree rooted at ROOT in level order by destructively
     *  modifying its .next pointers.  This tree must be a complete
     *  binary tree, with its next pointers initially null.  */
    static void linkInLevelOrder(BinTree root) {
        BinTree firstOfRow;
        firstOfRow = root;
        for (BinTree node = firstOfRow; node != null; node = node.next) {
            if (node.left != null && node.right != null) {
                node.left.next = node.right;
                if (node.next != null) {
                    node.right.next = node.next.left;
                }
            }
            if (node.next == null) {
                node.next = firstOfRow.left;
                firstOfRow = node.next;
            }
        }
    }
}
```

3. [2 points] We are given the following array representing a min-heap containing seven values, where each letter represents a different number. The last array position is initially unused. (For convenience, we've numbered the array elements starting from 1 rather than 0.)

1	*2*	3	*4*	*5*	*6*	*7*	*8*
A	B	C	D	E	F	G	

Given this initial min-heap, assume we insert a new value, H, also different from the rest, and restore the heap property. Assume that we use the heap-manipulation algorithms presented in lecture.

(a) Where in the array (at what indices) might H end up? (Fill in the squares for all that apply.)

[✓ 1] [✓ 2] 3 □ [✓ 4] 5 □ 6 □ 7 □ [✓ 8]

(b) Which letter(s) could represent the contents of array element 2 in the resulting min-heap?

[✓ A] [✓ B] C □ D □ E □ F □ G □ [✓ H]

(c) Which indices in the resulting array might contain the third smallest value?

1 □ [✓ 2] [✓ 3] [✓ 4] [✓ 5] [✓ 6] [✓ 7] 8 □

Now return to the original heap (just A–G; no H inserted). After removing the minimum value from the original heap and restoring the heap property:

(d) Which indices might contain the value G in the resulting min-heap?

1 □ [✓ 2] [✓ 3] [✓ 4] [✓ 5] [✓ 6] 7 □

(e) Which indices might contain the value B in the resulting min-heap?

[✓ 1] [✓ 2] 3 □ 4 □ 5 □ 6 □ 7 □

(f) Which letter(s) could represent the contents of element 4 in the resulting min-heap?

A □ B □ C □ [✓ D] E □ F □ [✓ G]

4. [3 points] Fill in the method `numberInRange` to fulfill its comment. The method must not look at any nodes unnecessarily;

```
class BST {
    public BST left, right;
    public int key;
}

class BSTUtils {

    /** Assuming T is a binary search tree, return the number of keys
     *  in T that are between L and H, inclusive. */
    public static int numberInRange(BST T, int L, int H) {
        if (T == null) {
            return 0;
        } else if (T.key < L) {
            return numberInRange(T.right, L, H);
        } else if (T.key > H) {
            return numberInRange(T.left, L, H);
        } else {
            return 1 + numberInRange(T.left, L, H)
                + numberInRange(T.right, L, H);
        }
    }
}
```

5. [2 points] Fill in the following function to agree with its comment. [Language Note: In Java, the notation `0bdddddddddd...`, where the 'd's are 1's and 0's, denotes a binary number. Java also allows the addition of underscores in binary and other numerals to break them up and make them more readable (examples: `15_234`, `0b1110_0001`). They have no effect on the numerical value.]

```
/** Returns the integer formed by repeating bits 31, 27, 23,
 *  19,..., 3 of X (where bit 0 is the rightmost (units) bit)
 *  in each of the three bits immediately to their right.
 *  For example,
 *     dup4(0b1101_0011_1111_1000_0110_1110_0111_0000)
 *       == 0b1111_0000_1111_1111_0000_1111_0000_0000
 */
static int dup4(int x) {
    int every4th = 0x88888888 & x;
    return every4th | (every4th >> 1) | (every4th >> 2) | (every4th >> 3)
}
```

6. [1 point] Where was the mnemonic phrase "Royal Flags Wave Kings Above" used?
Bletchley Park. The first letters R, F, W, K, A are the letters after the notches on rotors I, II, III, IV, and V of the Enigma machine.

7. [3 points] For each of the following methods, write down the tightest asymptotic $\Theta(\cdot)$ bounds you can for their worst-case and the best-case asymptotic running time as a function of $n \geq 0$. Make your bounds as simple as possible, with no unnecessary terms or constants. Assume that all other functions called by each method take constant time.

(a)

```
public void oneFunc(int n) {

    for (int i = n; i < n * n; i += 1) {
        doSomething(i);
    }
}
```

Worst Case: $\Theta(n^2)$

Best Case: $\Theta(n^2)$

(b)

```
public void twoFunc(int n) {

    for (int i = 0; i < n; i += 1) {
        for (int j = i; j < n - i; j += 1) {
            if (p(i, j, n)) {
               break;
            }
        }
    }
}
```

Worst Case: $\Theta(n^2)$

Best Case: $\Theta(n)$

(c)

```
public void redFunc(int n) {

    if (n > 0) {
        redFunc(n - 1);
        for (int i = 0; i < n; i += 1) {
            if (p(i, n)) {
                break;
            }
            h(i, n);
        }
    }
}
```

Worst Case: $\Theta(n^2)$

Best Case: $\Theta(n)$

(d)
```
public void blueFunc(int n)

    if (n > 0) {
        blueFunc(n / 2);
        for (int i = 0; i < n; i += 1) {
            if (p(i, n)) {
                break;
            }
            h(i, n);
        }
    }
}
```

Worst Case: $\Theta(n)$

Best Case: $\Theta(\lg n)$

(e)
```
public void myFunc(int n) {

    for (int i = 0; i < n; i += 1) {
        for (int j = 1; j < i; j *= 2) {
            h(i, j, n);
        }
    }
}
```

Worst Case: $\Theta(n \lg n)$

Best Case: $\Theta(n \lg n)$

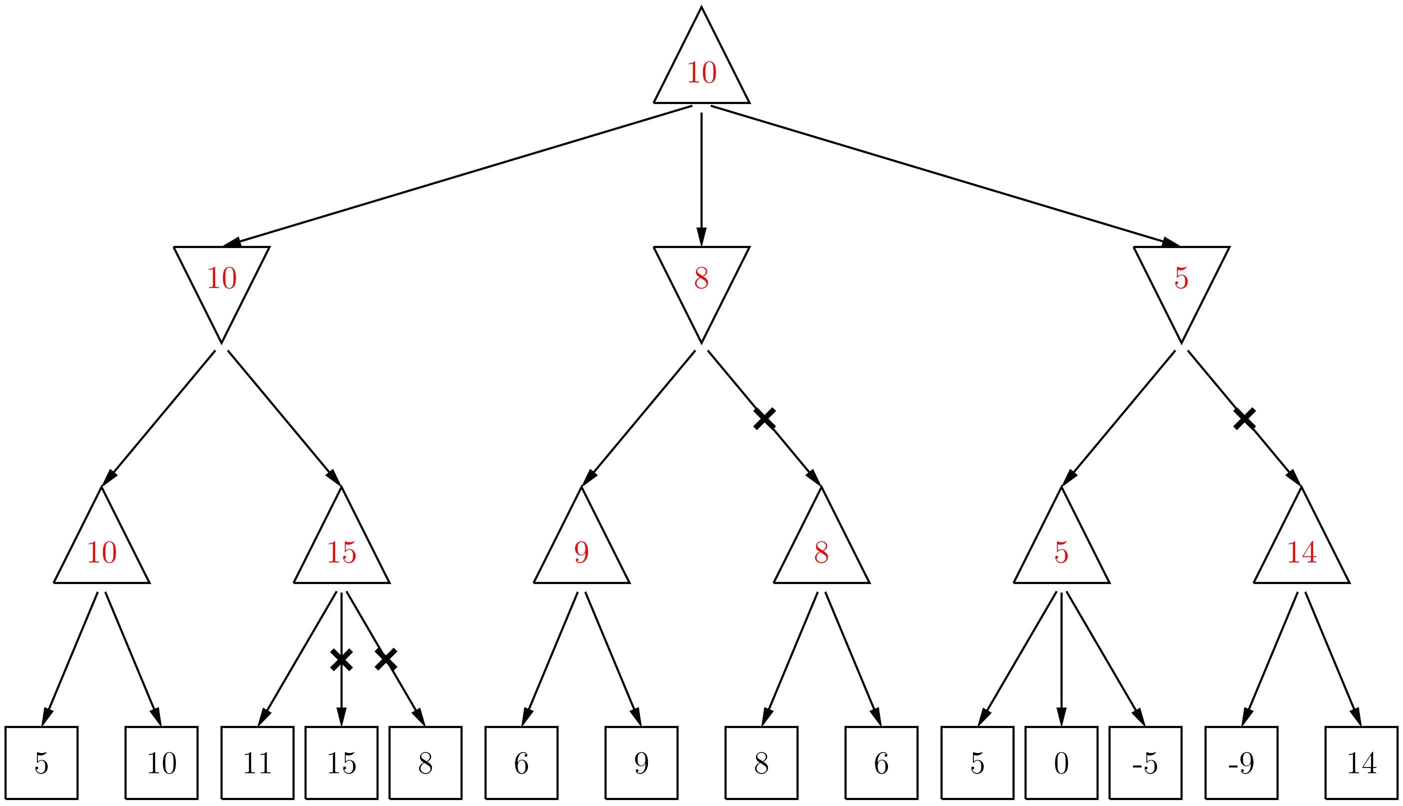
(f)
```
public void yourFunc(int n) {

    if (n < 1) {
        return;
    } else {
        yourFunc(n / 2);
        if (g(n)) {
            for (int i = 0; i < n; i++) {
                h(i, n);
            }
            yourFunc(n / 2);
        }
    }
}
```

Worst Case: $\Theta(n \lg n)$

Best Case: $\Theta(\lg n)$

8. [2 points] In the partial game tree below, we represent maximizing nodes as △; minimizing nodes as ▽; and nodes with static values as □. Determine the values for the nodes that would be determined by the minimax algorithm without pruning (write them inside the nodes), and then cross out branches that would not be traversed (would be pruned) as a result of alpha-beta pruning. Assume we evaluate children of a node from left to right.


10
10
8
5
10
15
9
8
5
14
5
10
11
15
8
6
9
8
6
5
0
-5
-9
14