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

CS61B **P. N. Hilfinger**
Spring 2020

Test #2

READ THIS PAGE FIRST. *Please do not discuss this exam with people who haven't taken it.* Your exam should contain 7 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. [3 points] In the following, give the tightest bounds you can for each part ($\Theta(\cdot)$ if possible, and otherwise the smallest $O(\cdot)$ and largest $\Omega(\cdot)$ you can).

(a) Consider a Java `HashSet H` that contains a certain object (call it `obj`). I wish to execute the ***destructive*** method `obj.mogrify()` while `obj` is in `H`. If I simply call `obj.mogrify()`, what could go wrong? Be specific as to what operations on `H` might fail as a result.

Since `H`'s hash value might change (the operation is destructive), `H` might now be in the wrong bucket. A future look-up operation could therefore fail and a future `add` operation could cause `H` to appear multiple times.

(b) In light of part (a), what should we do when modifying `obj` so that the failing operations you listed in part (a) won't fail?

First remove `obj` from `H`, modify it, and then re-insert it.

(c) Consider a hash table with external chaining that can contain integers in some range $0 \leq i < N$ and that has M buckets, where $M^2 > N$. The table uses the hash function $h(x) = \lfloor x/(N/M) \rfloor$ ($\lfloor y \rfloor$ is the largest integer not exceeding y), and is never resized. Suppose that we have a set of integers S such that any two members of S differ by at least $M/4$. What bound can we place on the worst-case time of adding all the integers in S to an initially empty table and then performing M look-ups on the table? (Hint: by the conditions of the problem, the size of S is at most $N/(M/4) = 4(N/M) < 4M$.)

$\Theta(M)$. There can be at most 4 items per bucket.

(d) Under the same conditions as in part (c), but with the hash function $h(x) = x \bmod M$?

$\Theta(M^2)$ or $\Theta(N)$. Nothing prevents all items going into the same bucket in this case. Even if one doesn't check for duplicates (during addition), the total lookup time will come out to M searches through M items in the bucket.

(e) Consider S, N, and M as in part (c), with $h(x) = \lfloor 8x/(N/M) \rfloor$ in an open-address hash table with $8M$ buckets and linear probes (+1, +2, etc.). As before, what bound can we place on the worst-case time of adding all the integers in S to an initially empty table and then performing M look-ups on the table?

$\Theta(M)$. There will be no collisions in this table, because adjacent elements of S must be $M/4$ apart, so that their hash values must be $8(M/4)/(N/M) = 2M^2/N > 2$ apart. Since elements are separated by at least one empty element, lookups must end after one test.

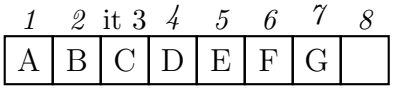
(f) One idea for improving the speed of an externally chained hash table when there might be lots of collisions is to replace the linked lists for each bucket with a binary-search tree. Suppose we use the algorithms from class for binary search trees and

insert a succession of P integers into the table ***in ascending order.*** Might this improve the total running time of these P operations relative to using linked lists for the buckets?

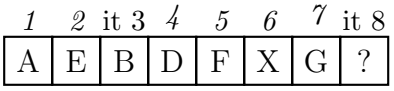
No. When keys are inserted in ascending order according to the basic insertion algorithms we've used so far, a binary search tree essentially becomes a linked list anyway.

2. [2 points]

We are given the following array representing a min-heap containing seven values (no duplicates), where each letter represents a distinct number. The last array position is initially unused. (For convenience, we've numbered the array elements starting from 1 rather than 0.)



Given this initial min-heap, we perform a sequence of four operations, each of which either adds a value or removes the minimum value. We never add a value when it is in the heap, although we may re-add a value after it has previously been removed. Assume we use the heap-manipulation algorithms presented in lecture. We end up with following array. The value X again differs from A–G, and '?' indicates an unknown value (in fact, that array element might not have been changed.)



a. Determine a sequence of four operations ("`add(`*value*`)`" or "`removeMin()`") that gives this result.

`removeMin()` _____

`removeMin()` _____

`add(X)` _____

`add(A)` _____

b. Fill in the following comparisons with either `>`, `<`, or `?` to indicate what you can deduce about the relative priorities of the values (`?` means it can't be determined).

`D` `>` _____ `E`

`X` `>` _____ `G`

`B` `>` _____ `C`

`G` `<` _____ `F`

3. [5 points] Define a binary-tree node as follows:

```
class BinTree {
    public BinTree left, right;
    public int key;

}
```

and define the following abstract class:

```
abstract class BinTraveler {
    /* doLeft, doRight, and doHere do not change tree or this BinTraveler. */
    abstract boolean doLeft(BinTree tree);
    abstract boolean doRight(BinTree tree);
    abstract boolean doHere(BinTree tree);
    abstract void visit(BinTree tree);

    void process(BinTree tree) {
        if (doLeft(tree)) {
            process(tree.left);
        }
        if (doHere(tree)) {
            visit(tree);
        }
        if (doRight(tree)) {
            process(tree.right);
        }
    }
}
```

Fill in the function below and the class `Lister` on the next page to fulfill the comments. The call `valuesLess(tr, k)` must produce its result in time $O(h + M)$, where h is the height of `tr` and M is the size of the result of the call.

```
/** Assuming that TREE is a binary search tree with
 *  no duplicate values, return a list of all keys in it
 *  that are strictly less than LIMIT, in order. */
static ArrayList<Integer> valuesLess(BinTree tree, int limit) {
    Lister L = new Lister(limit);
    L.process(tree);
    return L.list();
}
```

```
class Lister extends BinTraveler {
    Lister(int limit) {
        _limit = limit;
    }

    @Override
    boolean doLeft(BinTree tree) {
        return tree != null;
    }

    @Override
    boolean doRight(BinTree tree) {
        return tree != null && tree.key < _limit;
    }

    @Override
    boolean doHere(BinTree tree) {
        return tree != null && tree.key < _limit;
    }

    @Override
    void visit(BinTree tree) {
        _result.add(tree.key);
    }

    ArrayList<Integer> list() {
        return _result;
    }

    private int _limit;
    private ArrayList<Integer> _result = new ArrayList<>();
}
```

4. [3 points] In the following, use only the bit operators `&`, `|`, `^`, `~`, `<<`, `>>`, `>>>`. [Language Note: In Java, the notation `0bddddddddd...`, 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.]

(a) Fill in the following function to agree with its comment.

```
/** Return the int R in which for each k, 0 <= k < 32:
 *     + If bit k of C is 0, bit k of R is the same as bit k of A, and
 *     + If bit k of C is 1, bit k of R is the same as bit k of B.
 *
 *  For example,
 *
 *     mix(0b1011011011,
 *         0b1000110110,
 *         0b0101010101)
 *      == 0b1010011110
 */
static int mix(int a, int b, int c) {
    return (c & b) | (~c & a);
}
```

(b) Assuming a correct implementation of `mix` from part (a), what is a short, equivalent way of writing the expression

```
mix(x, ~x, y)
```

(i.e., using no function calls and as few bit operators as possible?

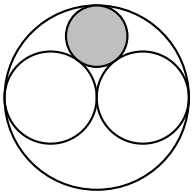
`x ^ y`

(c) Fill in the following function to agree with its comment.

```
/** Return the int in which each individual byte of N has been
 *  rotated left by 1. For example,
 *     rotBytes(0b10111010_00110011_01010101_10000000)
 *   =          0b01110101_01100110_10101010_00000001
 */
static int rotBytes(int n) {
    int M1 = 0x80808080;

    return ((n & ~M1) << 1) | ((n & M1) >>> 7);
}
```

5. [1 point] In the diagram below, what is the radius of the shaded circle, given that the larger circle has radius 30 and the other two each has radius 15 (drawing not necessarily to scale)?



Answer: 10. You can figure this out pretty directly, since it is a special case, or use Descartes' Theorem, which in verse goes [From *Nature*, June 20, 1936]:

> . . .
> Four circles to the kissing come,
> The smaller are the benter.
> The bend is just the inverse of
> The distance to the center.
> Though their intrigue left Euclid dumb
> There's now no need for rule of thumb.
> Since zero bend's a dead straight line,
> And concave bends have minus sign,
> The sum of the squares of all four bends
> Is half the square of their sum. . . .

6. [3 points] For each of the following methods, write down the tightest asymptotic $\Theta(\cdot)$ bounds you can for its 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, and that any upper-case single-character identifier (such as `M`) is an arbitrary positive constant.

(a)
```
public void funcOne(int n) {
    for (int i = 0; i < n; i += 1) {
        for (int j = i; j - i < M; j += 1) {
            if (test(j)) {
                doSomething(i, j);
            }
        }
    }
}
```

Best Case: $\Theta(n)$

Worst Case: $\Theta(n)$

(b)
```
public void funcTwo(int n) {
    for (int i = 0; i < n; i += 1) {
        for (int j = i; j - i < n/M; j += 1) {
            if (test(j)) {
                doSomething();
                break;
            }
        }
    }
}
```

Best Case: $\Theta(n)$

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

(c)
```
public void funcThree(int n) {
    int j;
    j = 0;
    for (int i = 0; i < n; i += M) {
        for (j = j+1; j < i + M; j += 1) {
            if (test(j)) {
                doSomething();
                break;
```

Best Case: $\Theta(n)$

Worst Case: $\Theta(n)$

```
            }
        }
    }
```

(d)
```
public void funcFour(int n) {
    if (n < M) {
        doSomething();
    } else if (test(n)) {
        funcFour(n - M);
        funcFour(n - M);
    } else {
        funcFour(n / M);
        funcFour(n / M);
    }
}
```

Best Case: $\Theta(n)$

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

(e) For the following, determine the running time for a call `funcFive(n, true)`.

```
public void funcFive(int n, boolean which) {
    if (n < 1) {
        doSomething();
    } else if (which) {
        funcFive(n - 1, true);
        funcFive(n, false);
    } else {
        funcFive(n - 1, false);
    }
}
```

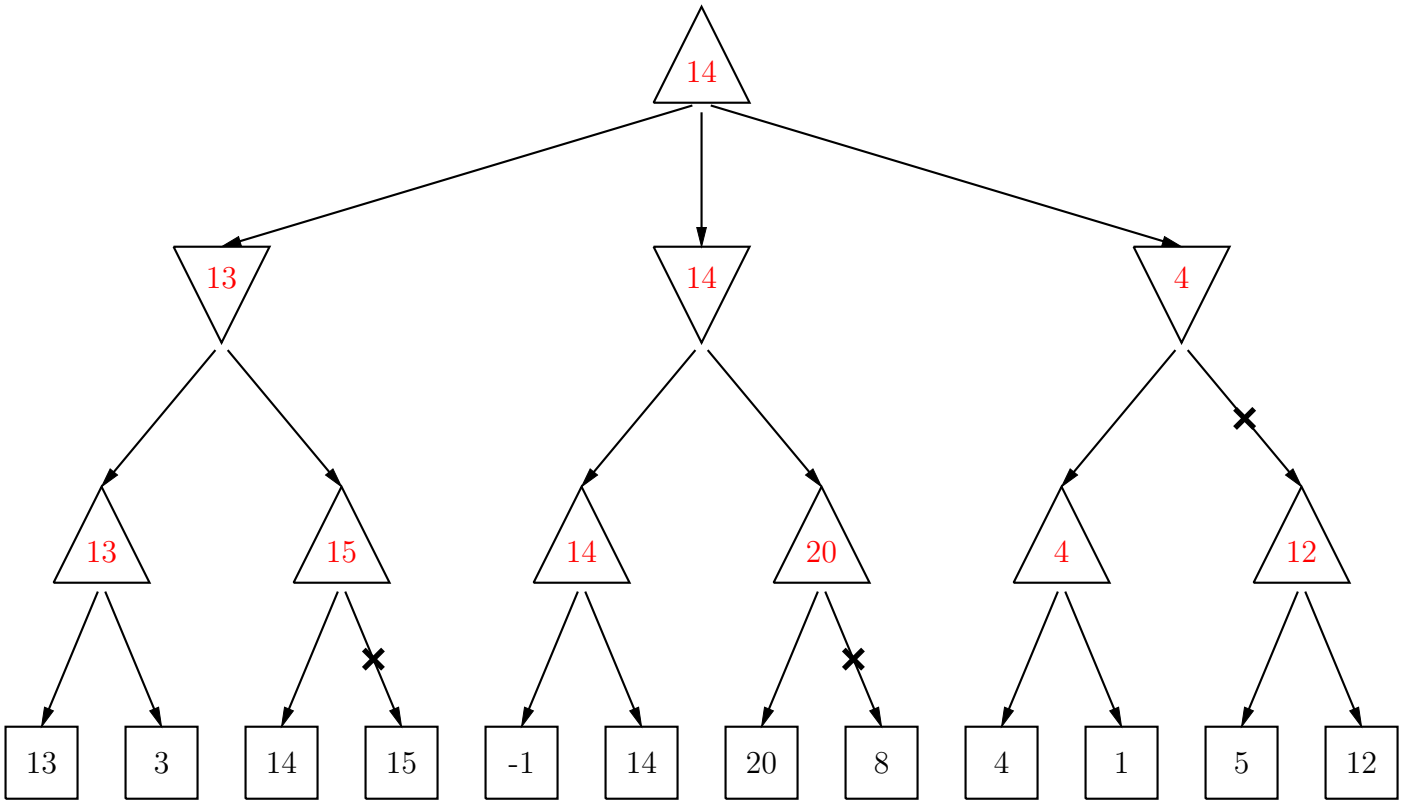
Best Case: $\Theta(n^2)$

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

(f) Throughout this course (and in particular for the preceding parts of this problem), we have analyzed complexity questions as if we were dealing with the standard set of all the mathematical integers, commonly written as $\mathbb{Z}$. Suppose instead we stick to actual Java `ints` and `longs`. What is the proper worst-case answer to part (b) above, and why?

$\Theta(1)$, because `n` is bounded by the constant $2^{31} - 1$.

7. [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 (fill in the blanks lettered a–k), and indicate which edges (lettered A–T) 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, and that all values are integers. In one case, a branch has already been pruned for you and you must determine a value for the node labeled k that would cause that pruning, subject to the constraint that *the values of all children of a node must be different).* If a branch is pruned, don't bother to indicate that the branches below it are also pruned.



a. Values of a–k.

b. Edges A–T to be pruned.