CS61BL Midterm 1 Review

1 Grandpuppies

Consider the following implementation of `Dog` and answer the questions below.

```
public class Dog {

    public String name;

    public Dog(String name) {
        this.name = name;
    }

    public Dog giveBirth() {
        return new Dog(this.name + "'s puppy");
    }

    public void bark() {
        System.out.println(this.name + " barks!");
    }

    public static void main(String[] args) {
        Dog[] myDogs = new Dog[3];
    }
}
```

(a) Given the above code, what would you write in the main method to populate `myDogs` with 2 new Dogs named Fido and Fiddle?

(b) How would you make Fido's grandpuppy bark in only one line of code?

(c) What would your answer to (b) output?

(d) What would happen if we tried to call `myDogs[2].bark()`?

2 Animalistic

Fill in the blanks and cross out and rewrite lines of code in the `Animal` and `Dog` classes so that `Foo` compiles and prints out the following lines:

```
1
2
3
Superdog
Superdog
bark 3
4
```

Any crossed out line can only be replaced by one rewritten line, if necessary. Do not add new lines of code anywhere except in the blanks provided. Do not modify `Foo`.

```
package zoo;

public class Animal {

    int noise;
    private String name; // Do not modify this line

    public Animal(String name) {
        name = name;
    }

    public void makeNoise() {
        ____________________________;
        System.out.println(_____________);
    }

    public void sayName() {
        System.out.println(name);
    }

    ____________________________________
}
```

```
package housepets;
import zoo.Animal;

public class Dog {

    public Dog() {
        ____________________________;
    }

    public void makeNoise(String sound) {
        System.out.println(sound + " " + _____________);
    }
}
```

`Foo` *class and* `main` *method on the next page.*

```
import zoo.Animal;
import housepets.Dog;

public class Foo { // Do not modify class Foo

    public static void main(String[] args) {
        Animal a = new Dog();
        Animal b = new Dog();
        Animal c = new Dog();
        a.makeNoise();
        b.makeNoise();
        a.makeNoise();
        c.sayName();
        a.sayName();
        a.makeNoise("bark");
        c.makeNoise();
    }
}
```

Expected output reproduced below.

```
1
2
3
Superdog
Superdog
bark 3
4
```

3 Strange

Fill in the `strangeInsertFront` method below such that both print statements at the bottom of this page print out `0 1 2 3 4`.

```
public class IntList {

    public int item;
    public IntList next;

    public IntList(int item, IntList next) {
        this.item = item;
        this.next = next;
    }

    public void print() {
        System.out.print(val + " ");
        if (next != null) {
            next.print();
        }
    }

    public IntList strangeInsertFront(int x) {
        ----------------------------------------------;
        ----------------------------------------------;
        return this;
    }
}

public class P2 {

    public static void main(String[] args) {
        IntList L = new IntList(4, null);
        L = L.strangeInsertFront(3);
        L = L.strangeInsertFront(2);
        L = L.strangeInsertFront(1);

        IntList L2 = L;
        L = L.strangeInsertFront(0);

        L.print();  // 0 1 2 3 4
        L2.print(); // 0 1 2 3 4
    }
}
```

4 Except Me for Who I Am

Consider the following implementation of `IntList` and answer the questions below.

```
public class IntList {

    public int item;
    public IntList next;

    public int getIndex(int item) {
        int index = 0;
        IntList temp = this;
        while (temp.item != item) {
            temp = temp.next;
            index += 1;
        }
        return index;
    }
}
```

(a) What happens when you call `getIndex(int item)` on an element that is not in the list?

(b) Write `getIndexExcp`, which attempts to get the index of an item, but throws an `IllegalArgumentException` with a useful message if no such item exists in the list. Do not use `if` statements, `while` loops, `for` loops, or recursion. You may use `getIndex`.

```
public int getIndexExcp(int item) throws IllegalArgumentException {









    }
```

(c) Write `getIndexNeg`, which attempts to get the index of an item, but returns -1 if no such item exists in the list. Do not use `if` statements, `while` loops, `for` loops, or recursion.

```
public int getIndexNeg(int item) {









    }
```

5 logSwap

`logSwap` is a static method that takes as arguments an array of integers and an index into that array. It takes the value at the specified index, v, and swaps it with whatever value, h, is at half that index (rounded down, as for Java integer division), if $v < h$. If the swap occurs, it then repeats the entire process, starting at the new (halved) index of v. If `logSwap` receives a starting index that is out of the array's bounds, `logSwap` throws an `IllegalArgumentException`.

```
/** Swap the value at ARR[INDEX] down the array (towards index 0),
 * starting with the element halfway between INDEX and 0 and
 * continuing to reduce the destination index by 1/2 as long as the
 * value there is strictly smaller. Throws
 * IllegalArgumentException if INDEX is out of bounds.
 *
 * For example, if A initially contains { 7, 6, 5, 4, 3, 2, 1 },
 * then after logSwap(A, 6), it contains { 1, 7, 5, 6, 3, 2, 4 }.
 * If B initially contains { 7, 1, 5, 4, 3, 2, 1 }, then after
 * logSwap(B, 5) it contains { 7, 1, 2, 4, 3, 5, 1 }.
 */
public static void logSwap(int[] arr, int index) {









}
```

6 wordCount

Determine the asymptotic runtime bound for `wordCount` in terms of N, the length of the array, `words`. Assume that all of the Strings in `words` have some length bounded by a constant.

Give your solution in terms of O, Ω, or Θ. You may assume that `Arrays.sort` is in $\Theta(N \log N)$.

```
import java.util.Arrays;

public class Asymptotics {

    public static int wordCount(String[] words) {
        Arrays.sort(words);
        int N = words.length;
        int wordCount = 0;
        int i = 0;
        while (i < N) {
            String thisWord = words[i];
            wordCount += 1;
            int j = i + 1;
            while (j < N) {
                if (!words[j].equals(thisWord)) {
                    break;
                }
                j++;
            }
            i = j;
        }
        return wordCount;
    }
}
```

Bound: _____

(a) Suppose that we replace `Arrays.sort` with another sorting method that is $\Theta(N)$. Does this affect the runtime of `wordCount`? If so, what is the new runtime bound? Explain your answer.

(b) Suppose that we replace `Arrays.sort` with another sorting method that is $\Theta(N^3)$. Does this affect the runtime of `wordCount`? If so, what is the new runtime bound? Explain your answer.

7 Asymptotics Potpourri

For each of the questions below, give the asymptotic runtime bound as a function of the given parameters. Your answer should be simple, with no unnecessary leading constants or summations.

```
public static void counter(int n) {
    for (int outer = 1; outer < n; outer *= 2) {
        int inner = 0;
        while (inner < n) {
            inner++;
        }
    }
}
```

Bound: _____

```
public static int duplicates(int[] A) {
    int N = A.length;
    int S = 0;
    for (int i = 0; i < N; i += 1) {
        for (int j = i+1; j < N; j += 1) {
            if (A[j] == A[i]) {
                S += 1;
                break;
            }
        }
    }
    return S;
}
```

Bound: _____

```
public static int sumAfterPos(int[] A) {
    int N = A.length;
    for (int i = 0; i < N; i += 1) {
        if (A[i] > 0) {
            int S = 0;
            for (int j = i + 1; j < N; j += 1) {
                S += A[j];
            }
            return S;
        }
    }
    return 0;
}
```

Bound: _____

8 Challenge: Simple as A, B, C

Cross out any lines that cause compiler or runtime errors. What does the `main` program output after removing those lines?

```
class A {
    int x = 5;
    public void m1() { System.out.println("Am1->" + x); }
    public void m2() { System.out.println("Am2->" + this.x); }
    public void update() { x = 99; }
}

class B extends A {
    int x = 10;
    public void m2() { System.out.println("Bm2->" + x); }
    public void m3() { System.out.println("Bm3->" + super.x); }
    public void m4() { System.out.print("Bm4->"); super.m2(); }
}

class C extends B {
    int y = x + 1;
    public void m2() { System.out.println("Cm2->" + super.x); }
    public void m3() { System.out.println("Cm3->" + super.super.x); }
    public void m4() { System.out.println("Cm4->" + y); }
    public void m5() { System.out.println("Cm5->" + super.y); }
}

class D {
    public static void main (String[] args) {
        A b0 = new B();
        System.out.println(b0.x);
        b0.m1();
        b0.m2();
        b0.m3();

        B b1 = new B();
        b1.m3();
        b1.m4();

        A c0 = new C();
        c0.m1();

        A a1 = (A) c0;
        C c2 = (C) a1;
        c2.m4();
        ((C) c0).m3();

        b0.update();
        b0.m1();
    }
}
```