CS 61B Small Group Tutoring
Summer 2020 Section 4: Dynamic Method Selection, Inheritance, and Asymptotics

Worksheet 4

1 List'em all!

List all the asymptotic runtimes from quickest to slowest.
$\theta(n^2), \theta(n^{0.5}), \theta(\log n), \theta(3^n), \theta(c), \theta(n^{n!})\theta(n), \theta(n \log n), \theta(n!), \theta(n^n), \theta(2^n)$

2 What's that runtime?

For each of the methods below, please specify the runtime in BigO, BigΘ or BigΩ Notation. Please give the tightest bound possible.

```
_________ private static void f(int n) {
              for(int i = 0; i < n; i++) {
                  for(int j = 0; j < n; j++) {
                      linear(n); // runs in linear time with respect to input
                  }
              }
          }

_________ private static void g(int n) {
               if (n < 1) return;
               for(int i = 0; i < n; i++) {
                  linear(100);
               }
               g(n/2);
               g(n/2);
           }

_________ private static void h(int n) {
               Random generator = new Random();
               for(int i = 0; i < n; i++) {
                  if(generator.nextBoolean()) {
                      /* nextBoolean returns true with
                         probability .5. */
                      break;
                  }
               }
           }
```

3 How fast?

Given an IntList of length N, provide the runtime bound for each operation. Recall that IntList is the naive linked list implementation from class.

```
public class IntList {
   int item;
   IntList next;
}
```

Operations	Runtime
size()	
get(int index)	
addFirst(E e)	
addLast(E e)	
remove(int index)	
remove(Node n)	

4 The ABCs of OOP

Indicate what each line the main program in class **D** would print, if the line prints anything. If any lines error out, identify the errors as compile-time or runtime errors and cross out the corresponding lines.

```
public class A {
    public void x() { System.out.println("Ax"); }
    public void y(A z) { System.out.println("Ay"); }
}

public class B extends A {
    public void y() { System.out.println("By"); }
    public void y(B z) { System.out.println("Byz"); }
}

public class C extends A {
    public void x() { System.out.println("Cx"); }
}

public class D {
    public static void main(String[] args) {
        A e = new B();
        A f = new C();
        B g = new A();
        B h = new C();
        C i = (C) new A();
        B j = (A) new C();
        B k = (B) e;

        f.x();
        e.x();
        e.y();
        (B) e.y();
        ((B) e).y();
        e.y(e);
        e.y(f);
}
```

5 Classy Cats

Look at the `Animal` class defined below.

```
public class Animal {
    protected String name, noise;
    protected int age;

    public Animal(String name, int age) {
        this.name = name;
        this.age = age;
        this.noise = "Huh?";
    }

    public String makeNoise() {
        if (age < 2) {
            return noise.toUpperCase();
        }
        return noise;
    }

    public String greet() {
        return name + ": " + makeNoise();
    }
}
```

(a) Given the `Animal` class, fill in the definition of the `Cat` class so that it makes a ”Meow!” noise when `greet()` is called. Assume this noise is all caps for kittens, i.e. `Cats` that are less than 2 years old.

```
public class Cat extends Animal {















}
```

(b) ”Animal” is an extremely broad classification, so it doesn’t really make sense to have it be a class. Look at the new definition of the `Animal` class below.

```
public abstract class Animal {
    protected String name;
    protected String noise = "Huh?";
    protected int age;

    public String makeNoise() {
        if (age < 2) {
            return noise.toUpperCase();
        }
        return noise;
    }

    public String greet() {
        return name + ": " + makeNoise();
    }

    public abstract void shout();
    abstract void count(int x);
}
```

Fill out the `Cat` class again below to allow it to be compatible with `Animal` (which is now an abstract class) and its two new methods.

```
public class Cat extends Animal {
    public Cat() {
        this.name = "Kitty";
        this.age = 1;
        this.noise = "Meow!";
    }

    public Cat(String name, int age) {
        this();
        this.name = name;
        this.age = age;
    }

    @Override
    ____________ ____________ shout() {
        System.out.println(noise.toUpperCase());
    }

    @Override
    ____________ ____________ count(int x) {
        for (int i = 0; i < x; i++) {
            System.out.println(makeNoise());
        }
    }
}
```