**Midterm Solutions**

1. a) Look carefully: Is System.out.println() a classwide method or an instance method? Explain:

It is an instance method. System is a class that has a class field named out. The field System.out refers to an object. The object System.out has an instance method named println().

b) What’s log10(2.5) ?

log102 = (log101024) (log10242) ≈ 3 \* .1 ≈ .3

log105 = log10(10/2) = log1010 - log102 ≈ 1 - .3 ≈ .7

log102.5 = log10(5/2) = log105 - log102 ≈ .7 - .3 ≈.4

c) Characterize the recurrence

T(n) = if n = 1 then 50 else 16T(n/5) + 15n3 + 1

This is a divide and conquer recurrence.

f(n) = 15n3 + 1 ∈ Θ(n3)

a = 16 b = 5 d = 3

16 < 53

By Master Theorem, we find T(n) ∈ Θ(n3)

d) Characterize the recurrence

T(n) = if n = 1 then 1 else 8T(n/2) + 9n3 + 10n2 + 11n + 12

This is a divide and conquer recurrence.

f(n) = 9n3 + 10n2 + 11n + 12∈ Θ(n3)

a = 8 b = 2 d = 3

8 = 23

By Master Theorem, we find T(n) ∈ Θ(n3 log n)

e) What is garbage collection and how does it work in Java?

In Java, memory for objects is dynamically allocated from the heap. When an object is created, there is a variable pointing to that Object (i.e., that place in memory). When there are no longer any variable pointing to that object, that object becomes what is known as an orphaned object. Java then implicitly does garbage collection where is de-allocates the memory that was being used for the orphaned object. Java does this implicitly, unlike languages like C++. However, you can explicitly call the garbage collector with (Runtime.getRuntime()).gc() or System.gc().

Furthermore, the garbage collector runs in a separate thread and runs concurrently.

1. As succinctly as possible, explain the difference between the pairs of items
2. is-a vs. has-a

If a class is another class, then that class extends that other class. If a class has another class, then it contains an instance of that other class. Is-a means the class is a member/subclass of some super class, and thus inherits all the methods and public/protected fields of said super class. Has-a means that the class contains an instance of some class, but does not extend said class.

1. Interface vs. abstract class

An interface has absolutely no implementation whereas abstract class can contain some implementation. Neither can be instantiated. In Java, a class can only extend one other class whereas a class can implement as many interfaces as you would like. Both contain abstract methods (the methods in interfaces are implicitly abstract and it is convention to omit the abstract modifier). A concrete class can implement an interface or extend an abstract class if it provides implementation for all of the abstract methods in both cases. Also, interfaces guarantee certain behavior.

1. ArrayList vs. LinkedList

An ArrayList is a list using array-based(sequential) implementation and a LinkedList is a list using pointer-based(linked) implementation. With array-based implementation, worst-case for random access is Θ(1) because arrays are indexed. However, add methods can be worst-case of Θ(n) since if you are maxed out of space in memory, you must copy everything over. With linked-based implementation, random access is typically Θ(n) since it is not indexed and you must follow pointers to find an element. However, adding with linked-based implementation is always Θ(1) since you just have to change the head pointer to the new element and set the next pointer of the new element to the previous head.

1. Θ(n2) vs. Θ(2n)

Θ(n2) is polynomial and grows slower asymptotically that Θ(2n), which is exponential. As n approaches infinity, 2n > n2 (which can be proved via induction).

1. Big-oh vs. big-omega

Big-oh is the upper bound of a function. Therefore, f(n) ∈ O(g(n)) if there exist constants c and no  such that for all n ≥ no, f(n) ≤ c \* g(n).

Big-omega is the lower bound of a function. Therefore, f(n) ∈ \Omega (g(n)) if there exist constants c and no  such that for all n ≥ no, f(n) ≥ c \* g(n).

If f(n) ∈ O(g(n)) and f(n) ∈ \Omega (g(n)), then f(n) ∈ Θ (g(n)) and it is tightly bound.

1. Draw the “cloud diagram” that results from executing this code.

Dice d = new Dice[4];

d[0] = new Dice();

d[0].setValue(2);

d[1] = new Dice();

Dice.setValue(d[1],5);

d[2] = d[0];

d[3] = d[d[0].getValue()];

Dice d4 = d[1];

Dice object

int value

5

d4

d

Dice object

int value

2

b) complete signature of method used on line 3:

public void setValue(int i);

c) complete signature for method used on line 5:

public static void setValue (Dice d, int i);

d) complete signature of method used on line 7:

public int getValue();

f) Complete this method, which returns true if and only if Collection c contains string s:

static boolean containsString (String s, java.util.Collection c){

return c.contains(s);

}

OR (with iterator)

static boolean containsString(String s, java.util.Collection c){

for (Object o : c){

if (((String) o).equals(s)){

return true;

}

return false;

}

}

4.

this.st = st;

this.ptr = ptr;

return new Node(st, ptr);

//application code

Node first = Node.makeNew(“cat”, (Node.makeNew(“-12”, (Node.makeNew(“x”, null)));