Brennan Huber
HW6

1. In this case I would use a binary search tree. This is because we will need to data structure to be dynamic that way we can add as many circles as we need without having to resize. We also need the ability to traverse very quickly.

```
Public class Part1 {
	private class Circle {
	// this would be my private class circle that we will use to represent a circle
		int x;
		int y;
		int r;

		Circle(x, y, r){
			this.x = x;
			this.y = y;
			this.r = r;
		}
	}

	// collection of circles.
	Circle circles = new Circle[101];

	Part1() {
		// default constructor.
	}

	public void addCircle(int x, int y, int r) {
		for(int I = 0; I < 101; i++) {
			if(circles[I] == null) {
				circle[I] = new Circle(x, y, r);
			}
			break;
		}
	}

	public static boolean(Circle c, int x, int y) {
	// math formula to see if the point is inside of the circle.. Just a distance formula.
		return ((x – c.getX())*(x – c.getX()))+((y-c.getY())*(y-c.getY())) < (c.getR()*c.getR())
	}

	public static TreeNode containsPoints(int x, int y) {
		create a tree
		loop through collection of circles
			if(insideCircle(circle, x, y))
				insert that circle into tree

		return tree
	}
```

}

this algorithm (containsPoint)is Big-O(N), N being the number of circles in the total collection, because we have to check every circle in the collection to see if that circle possibly contains the point that was given.

2. This heap will have to hold 2 things, a reference to the list, and also an integer which holds the size of the list. So the heap will be organized by size (smallest being first in the heap). So to insert 1 visitor, we will retrieve the first item in the heap, go to the list, insert the visitor into the first position of the list, then finally you must check to make sure the heap is still in order. The time constraint for adding 1 item to the list will be Big-O(logN). This is because accessing the first item in the heap is O(1), adding 1 item to the list is O(1), but the percolate function is O(logN). So therefore worst case adding 1 person to the list will take logN time.

3. The best data structure to do this task would be a stack. You would just read in the input, scan the input for parenthesis, and depending upon whether it is an opening or closing you would push onto stack, or pop off the stack. The algorithm for this would be Big-O(N), this is because you have to read the input, scan the input, and then for each word push onto stack, then pop off the stack therefore it is linear.

```
// super generic psuedo code
Get user input

loop through input
        if input is an opening quotation
                push next word onto stack

        if input is a closing quotation
                pop off the stack and add it to the line to be printed

print line
```