Dungeon3D – a simple 3-dimensional dungeon generator
Language: Java
Generate Call:

```
Dunegon3D dungeonGenerator = new Dungeon3D(seed); // (int) seed is optional
ArrayList<Rooms> fullMap = dungeonGenerator.generate();
```

Part 1:

Define the data structure(s) that will be used to store the dungeon

The entire dungeon is stored as a link-list of Room objects, starting with the 'start' room each additional room is connected to up to 6 other room on the _connections array. Connections are stored on both sides of 2 rooms, allowing to load one room at a time during play without worrying about how big the dungeon is. Additionally, an ArrayList containing all rooms is present and used for element iteration and full dungeon map return. The total storage size is O(N) where N is the number of rooms stored, each room specifically contains:

A Room contains the following data structures:

```
int _x; // 4bytes
int _y; // 4bytes
int _z; // 4bytes
boolean _end; // VM-dependant, assumed 1byte
Room[] _connections = new Room[6]; // 4bytes per object/null reference = 6*4bytes = 24bytes
```

Total storage of a single room is 33bytes (with 1 byte boolean assumption).

We store _end, but there is no need to store _start because start is always at the origin (0,0,0) which is easy to check. Currently _end is set but un-used, it would be required on playthrough.

Part 2:

Define the function(s) that will generate a dungeon given the requirement for a dungeon and an initial generate method with a similar signature to:

```
var generate = function (int rooms) { … }
```

Note the above is written in JavaScript, but you should modify this to be stylized appropriately for your chosen programming language.

Please see attached dungeon3d.java file.

Part 3:

Indicate the runtime and memory complexity of your solution using Big O notation.

Memory complexity is defined above along with the data structure used to store the dungeon, memory usage is O(N) where N is the number of rooms, each room occupying 33bytes.

Runtime of the generate() function is O(N^2). The main limitting factor is the connectAllAdjacent() function, which needs to check all existing rooms for adjacency and connect them to current_room if adjacent. The findRoom(x,y,z) function is also limitting, however it could be easily optimized to O(1) runtime using lookup/hash tables, un-needed here because we are limited by connectAllAdjacent().