(a) a verbal description of your algorithm, (b) a corresponding pseudo code, (c) a sketch of a proof of correctness of your algorithm (i.e., an argument that your algorithm always produces the desired answer), (d) a tight running time estimate for your algorithm, and (e) a brief reasoning behind the running time estimate.

Planters

My code works by iteratively comparing the two largest members of the empty containers and the growing plants. If the largest empty container is larger than the largest growing plant, then both numbers are removed from their arrays, then the plant size is added to the empty array (representative of repotting the plant and using its old container). If, at any point, the largest empty container is smaller than the largest plant, there is no valid configuration. When the plants array has a size of zero the configuration is concluded to be valid, because every plant found a new home.

Pseudocode

```
do{
    biggestPlant = findLargestIndex(growingPlants);
    biggestPot = findLargestIndex(containers);

    if(biggestPot<biggestPlant){
        return "NO"
    }

    containers.add(growingPlant[biggestPlant])
    containers.rm(biggestPot)
    growingPlants.rm(biggestPlant)

}while( growingPlants.size>0 );
```

Proof of Correctness

Always comparing the largest of each has several effects:

1. Guarantees invalid configurations will be found
2. Reuses the largest plant container
3. Avoids false negatives

Essentially, if there is a time where no container can fill the next largest plant, the configuration is invalid.

Time Estimate

Finding the largest value in each array is n operations, and one less operation every iteration.
The amount of iterations is the amount of plants
(^ At worst for both measures)

I believe this makes my code upper bound by n^2 Estimate: $O(n^2)$