Jordan Allard
CSCI 264-01
Homework 1

HW1 Problem 3 Writeup

a) Verbal Description

Given p planters containing full planters of various sizes and r empty planters of various sizes, this algorithm determines if each of the plants in a full planter can be moved to an empty planter of a larger size. It does this by sorting the two arrays by size, then comparing each full planter against the largest value in the empty planter array. If the empty planter array has a maximum value larger than the full planter array, the empty planter size is swapped for the full planter size. If the empty planter maximum is equal to the full planter maximum, the algorithm compares against the next largest non-equal value. If at any point the algorithm finds that a value in the full planter array is larger than the maximum of the empty planter array, it returns NO.

b) Pseudocode

```
Let s be the set of full planters of size p
Let t be the set of empty planters of size r
Sort s and t from smallest to largest using MergeSort
Let duplicateOffset = 0
Let previousValue = 0
For every planter s[i] from i = p to i = 1: // counting backwards
      If s[i] == previousValue:
            duplicateOffset += 1
      Else:
            duplicateOffset = 0
      If duplicateOffset >= r or s[i] > t[r - duplicateOffset]:
            Print NO and return
      Else:
            t[r - duplicateOffset] = s[i]
Print YES and return
```

c) Proof of Correctness

The algorithm is correct because it checks every plant pot and removes plants from the array once they've been replanted, ensuring that there are no duplicate pots being used. Since both arrays are sorted from least to most value, it will always be the case that if the leftmost item in the empty pot array cannot hold the leftmost item from the planter array, there are no other pots in the empty array that can hold the plant, except when the planter array contains duplicate values. It is possible that the value of the new empty pot being swapped into the spot of the previous maximum pot is no longer the largest value in the empty pot array, but since the planter array is also sorted, it won't matter because any non-duplicate values are guaranteed to be smaller than the new value. Thus, the algorithm only must check the empty planter array for other possible maximums when duplicates occur, and since the array is still sorted, it need only check the leftmost non-duplicate value, since no other rightward values can be larger than it.

d) Running Time Estimate

O(n log n)

e) Running Time Estimate Reasoning

This algorithm uses a Merge Sort, which has time complexity of O(n log n). It first loops through each input, which is a time complexity of O(p) and O(r). It then runs Merge Sort exactly two times, the first with p elements O(p log p) and then with r elements O(r log r). Becase n = p + r, both p log p and r log r are O(n log n). The algorithm then runs a for loop exactly p times, so the final complexity

is O(p) + O(r) + O(p log p) + O(r log r) + O(p), which is equivalent to O(n log n).