A Prize No One Can Win - Recitation Write Up

Solution synopsis

First, let's cover a quick synopsis of the solution to the program. We are given a set of numbers and need to choose a subset of those numbers such that no two numbers in the subset add up to beyond some target value. Furthermore, we want to maximize the size of that subset.

Naturally, if we want numbers to add up to something small and we want as many of them as possible, our goal is to take the smaller numbers first. To do this, we must sort the input values.

Once we do that, we know we will select the values in indexes 0 through k, for some value k. Naturally, if one is trying to maximize the sum of two elements from this subset, they would choose array[k-1] and array[k].

So, we seek the maximum k such that array[k-1]+array[k] ≤ target.

Once the array is sorted, we just do a single for loop adding successive pairs of elements until the sum exceeds the target. (The choice before this is our answer.)

If all the values in the array exceed target, the answer is 1, which is what the initial answer should be set to, until valid values of k are found.

Problem with Posted Quick Sort

The posted quick sort uses a random partition element, which, as long as the items are ***distinct*** guarantees O(nlgn) performance. However, if the items are not distinct, and specifically, if all the items are the same, then this line of code in the partition:

```
while (low <= high && vals[low] <= vals[lowpos]) low++;
```

is guaranteed to move low all the way over past high, splitting the array into one array of size 0 and the other of size n-1, (if the original subarray size is n).

There's no real way to fix this within the partition code, without changing the items being sorted so that all items are distinct. (One could create a new struct that stores both the value and a unique ID number, breaking ties by that unique ID number.)

But, we notice that in this case, and perhaps some other cases, if the subarray is already sorted, there's no need to call quick sort AT ALL!!!

So, the fastest fix is simply to add an issorted function (which was in the sample code to begin with), and call it in the quicksort code as a base case:

```
if (issorted(numbers, low, high)) return;
```

The speed up is significant, because it quickly handles the key case that was producing the worst case performance of quick sort - when the input array was identical items.