Group 8

Thomas Dahlgren and Josh Quist

The algorithm

- We used bucket sort twice to sort the data
- The buckets were ArrayLists, and then the results are put back into the original array so it sorts in place
- The first time we sort the data into 24 buckets based on the numbers of 1's in the binary string of the integer, either has zero 1's or has at most 23 1's
- Then we sort each of those buckets into 13 buckets based on the longest repeating and non overlapping substring, as small as 0 or as big as 12
- As we added each element into the second set of buckets we used binary search to find the elements position based on its integer value, and then used arraylist.add(index, element)
- We did it this way because because it uses almost no comparisons and since the length of longest substring function takes the most time we wanted to have to call it the least amount of times

```
private static void sort(Integer[] toSort) {
    int n = toSort.length;
    if (n <= 0)
        return;
    ArrayList<ArrayList<Integer>> buckets = new ArrayList<>();
    for (int i = 0; i < 24; i++) {
        buckets.add(new ArrayList<Integer>());
    }
    for (int i = 0; i < n; i++) {
        int idx = Integer.bitCount(toSort[i]);
        buckets.get(idx).add(toSort[i]);
    }
    for (int i = 0; i < 24; i++) {
        buckets2(buckets.get(i));
    }
    int index = 0;
    for (int i = 0; i < 24; i++) {
        for (int j = 0; j < buckets.get(i).size(); j++) {
            toSort[index++] = buckets.get(i).get(j);
        }
    }
}
```

Length of longest repeating substring

- We used a modified version of code we found here: https://leetcode.com/problems/longest-duplicate-substring/discuss/1513594/Go-golang-shortest-solution
- This allowed for overlapping so we made changes to not allow for it
- It basically just checks if everything in the right substring is contained in the left substring and then increases the backIndex if it is
- Runs in m time since the while loop increases the index while it runs

```
public static int longestDupSubstringNoOverlap(String s){
    int max = 0;
    int backIndex = 1;
    int length = s.length();
    for (int i = 1; i < length; i++) {
        if(i + backIndex > length){
            return max;
        }
        else if (s.substring(0,i).contains(s.substring(i, i + backIndex))){
            while(i + backIndex <= s.length()
            && s.substring(0,i).contains(s.substring(i, i + backIndex))){
                max = backIndex;
                backIndex++;
            }
        }
    }
    return max;
}
```

10010 false
10010 true max++
10010 false
10010 true max++
Return 2

Efficiency

- The first bucket sort runs each loop 24 times, but runs 2 “for” loops n times so it runs in n time.
- Buckets2 runs in n(m+logn) time since it uses binarysearch(logn) and longestDupSubstring(m) each n times
- This makes the whole sorting algorithm run in n+n(m+logn) time which is O(nm) time. Since m is at most 25, and logn = 25 when n =10000000 and our n never got that big

```
private static void buckets2(ArrayList<Integer> toSort){
    int n = toSort.size();
    if (n <= 0)
        return;

    ArrayList<ArrayList<Integer>> buckets = new ArrayList<>();

    for (int i = 0; i < 13; i++) {
        buckets.add(new ArrayList<Integer>());
    }
    for (int i = 0; i < n; i++) {
        int idx = longestDupSubstringNoOverlap(Integer.toBinaryString(toSort.get(i)));
        int index = Collections.binarySearch(buckets.get(idx), toSort.get(i));
        if(index<0){
            index = index*(-1) - 1;
        }
        buckets.get(idx).add(index, toSort.get(i));
    }
    int index = 0;
    for (int i = 0; i < 13; i++) {
        for (int j = 0; j < buckets.get(i).size(); j++) {
            toSort.set(index++, buckets.get(i).get(j));
        }
    }
}
```

Results

Times for data set 1: 125, 133, 133 for third place overall

Times for data set 2: 176, 168, 189 for third place overall

The sum of medians is 309.0 for third place overall, first in class

We didn't get any correctness issues reported to us for our code