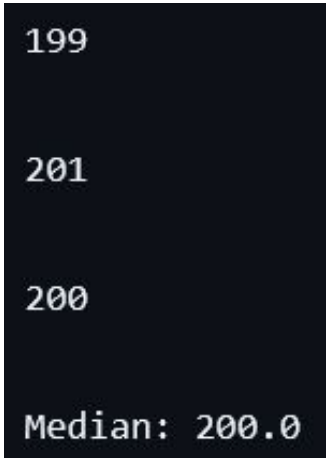
Group 4

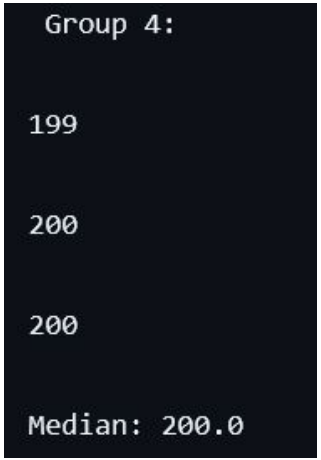
Sorting Competition Presentation

Anton Olson & James Swearingen

Final Times



Run 1



Run 2

Changes from Group 0

Modifying compare()

```
public int compare(String s1, String s2) {
    char[] c1 = s1.toCharArray();
    char[] c2 = s2.toCharArray();
```

Convert given strings into character arrays

```
    // if the two counts are the same
    int count1 = 0;
int count2 = 0;

for(int i = 0; i < target.length; ++i)
{
    if(target[i] != c1[i])
    {
        count1++;
    }
    if(target[i] != c2[i])
    {
        count2++;
    }
}

int count = count1 - count2;
```

Move distanceToTarget() to compare() and perform counting simultaneously

Changes from Group 0

Modifying compare()

```
if(count != 0)
{
    return count;
}
for(int i = 0; i < c1.length; i++)
{
    if(c1[i] > c2[i]) return 1;
    else if (c1[i] < c2[i]) return -1;
}
      return 0;
  }
```

Iterating linearly through each character of both strings:

1. Return “1” if String 1 is greater than String 2 (i.e., S1 = 1 and S2 = 2)
2. Return “0” if String 2 is greater than String 1
3. After iterating through both strings, if they’re equal, return 0

Changes from Group 0

Using Merge Sort

```
private static void sort(String[] toSort, String target) {
    // This, and all the methods/objects used here, can be changed/removed as you want.
    // You may modify the comparator, or not use it at all.
    // You may change the type of elements you are sorting and return an array of differen
    // type, as long as it has the same elements (just stored differently) in the same ord
    // as the sorted array of strings.

    // If you are creating or modifying any global variables, you need to reset them
    // to the original state after your sorting is done.

    SortingCompetitionComparator comparator = new SortingCompetitionComparator(target);


    mergeSort(toSort, comparator);
}
```

Merge Sort is a tad bit hard to fit onto a Google Slide, so I just put this here instead

We used Merge Sort as opposed to the built-in sorting method in Java because:

- **Stable**
- **Better on larger datasets**

(We used toSort[] to store our data)

Worst Case

O(L(nlogn))

Logic:

- Assume L = length of the string and n = number of elements
- We already know Merge Sort = O(nlogn)
- We need to iterate through L characters for all n to reach a comparison
- If all strings are the same, we'd be iterating through all of L in n strings, thereby making this the worse case