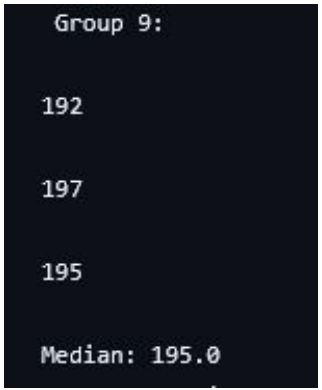
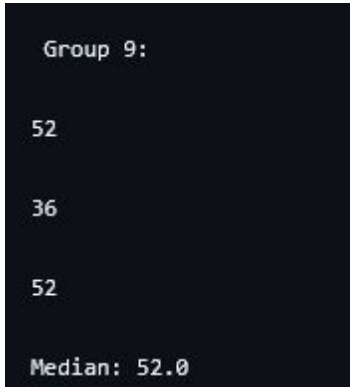
Group 9

Sorting Competition Presentation

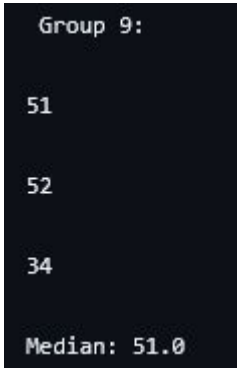
Anton Olson

Final Times





Run 1



Run 2

Changes from Group 0

Modifying sort()

```
private static void sort(String[] toSort, String target) {
    if (target.length() > 64)
        UInt128.sort(toSort, target);
    else
        LongAndString.shortSort(toSort, target);
}
```

If target string's length is > 64

UInt128.sort(toSort, target)

If target string's length is <= 64

LongAndString.shortSort(toSort, target)

Changes from Group 0

Adding UInt128.java

```java
public static void sort(String[] toSort, String target) {
    int n = toSort.length; // e.g. 7 million
    int width = target.length(); // e.g. 100

    @SuppressWarnings("unchecked")
    ArrayList<UInt128>[] array = new ArrayList[width + 1];

    // convert target into data type
    UInt128 convertedTarget = parseBinary128(target);

    Arrays.setAll(array, i -> new ArrayList<>(512));

    // the bitcount of XOR is the "distance" from target,
    // so for each value put it in arrayList dedicated for that distance
    for (int i = 0; i < n; i++) {
        UInt128 a = parseBinary128(toSort[i]);
        array[Long.bitCount(a.hi ^ convertedTarget.hi) + Long.bitCount(a.lo ^ convertedTarget.lo)].add(a);
    }
```

Process:

1. Take in toSort and target string
2. Convert target string and each toSort element in binary
    a. (stored in two halfs, both as longs)
3. Add toSort elements to newly created ArrayList array at the index of:
    a. The amount of 1s in the first half of the target string ^ (XOR) the first half of a toSort element + the amount of 1s in the first half of the target string ^ (XOR) the first half of a toSort element

Changes from Group 0

Adding UInt128.java

```
// Find the first non-empty bucket
int minIndex = 0;
while (minIndex < array.length && array[minIndex].isEmpty())
minIndex++;

// Find last non-empty bucket
int maxIndex = array.length - 1;
while (maxIndex >= 0 && array[maxIndex].isEmpty())
maxIndex--;

int putBackAfterSortIndex = 0;
// in each arraylist, sort the elements (null means use compareTo of elements)
// then put the values of the whole array (as a string) in its spot.
for (int i = minIndex; i <= maxIndex; i++) {
    array[i].sort(null);
    for (int j = 0; j < array[i].size(); j++) {
        toSort[putBackAfterSortIndex++] = array[i].get(j).raw;
    }
}
```

Process:

4. Find the first and the last non-null element in the ArrayList

5. Within the range of the first and last non-null element, sort the values using a comparator

6. The comparator compares the values using a method within the Long class called compareUnsigned()

Changes from Group 0

Adding LongAndString.java

```
static void shortSort(String[] toSort, String target) {

    int width = target.length();
    LongAndString convertedTarget = BinaryStringToLong(target);

    @SuppressWarnings("unchecked")
    ArrayList<LongAndString>[] array = new ArrayList[width + 1];
    Arrays.setAll(array, i -> new ArrayList<LongAndString>(512));
    Arrays.stream(toSort).map(raw -> BinaryStringToLong(raw))
            .forEach(a -> array[Long.bitCount(a.l ^ convertedTarget.l)].add(a));
    int index = 0;
    // in each arraylist, sort the elements (null means use compareTo of elements)
    // then put the values of the whole array (as a string) in its spot.

    for (int i = 0; i < width + 1; i++) {
        array[i].sort(null);
        for (int j = 0; j < array[i].size(); j++) {
            toSort[index++] = array[i].get(j).raw;
        }
    }
}
```

Process:

- Similar to UInt128

1. Convert target string and all strings within toSort[] into longs and add them to the index of an ArrayList array corresponding to the amount of 1s in the target string XOR each string of toSort[]
2. Sort the ArrayList array and use that to rearrange the toSort[] array

Worst Case

$$O(n*L + n\log n)$$

Logic:

- Since UInt128 deals with longer strings, it takes more time
- We already know Arrays.sort() utilizes O(nlogn) time due to utilizing TimSort
- We need to account for the parsing of n elements of l length into binary
    - O(n*L)
- Thus, O(n*L + nlogn)

Data Storage

UInt128

```
public UInt128(long h, long l, String r) {
    hi = h;
    lo = l;
    raw = r;
}
```

LongAndString

```
LongAndString(long l, String raw) {
    this.l = l;
    this.raw = raw;
}
```

ArrayList<>[] (in both)

```
ArrayList<UInt128>[] array = new ArrayList[width + 1];
```