Group 5 Sorting Algorithm

By Josh E and Josh E

Scores and Times

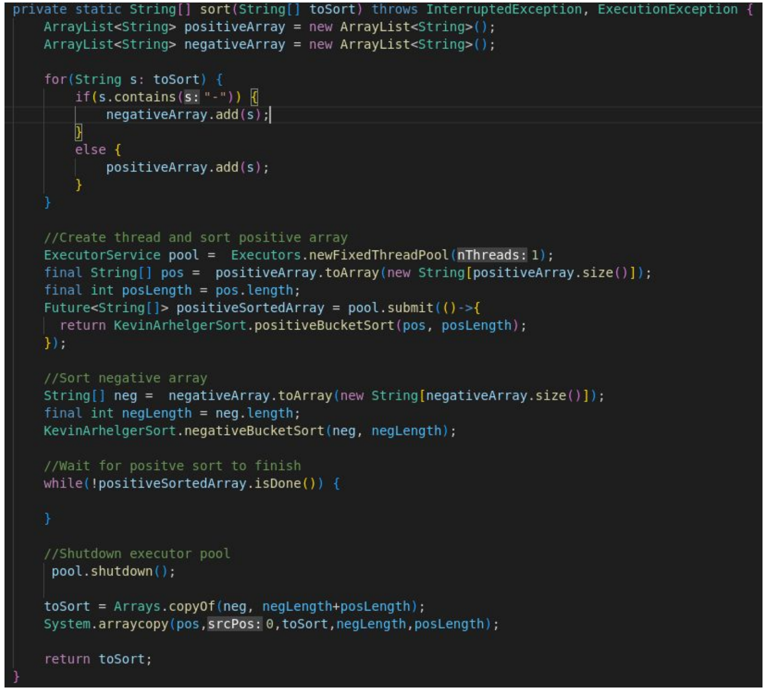
- Times
  - 1st dataset (51,234 elements)
    - 362 ms
    - 320 ms
    - 327 ms
  - 2nd dataset (700,000 elements)
    - 1989 ms
    - 1896 ms
    - 2245 ms
- Scores
  - 2nd overall
    - Sum of places was 4
    - Sum of medians was 2316.0 ms

Correctness discussion

- Correctness
  - No issues found
  - No rules were broken

Description of algorithm

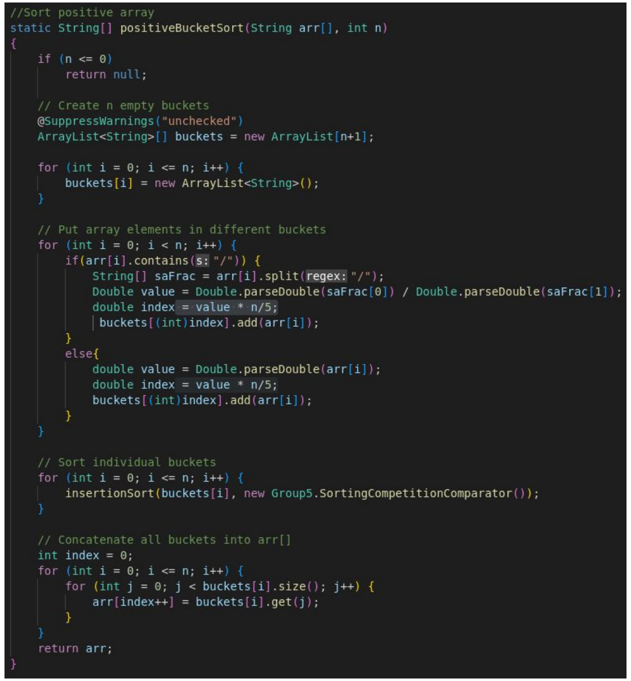
- Data structures used
  - Arrays
  - ArrayLists
- Main sort function
  - Create two ArrayLists
    - One for positive elements
    - One for negative elements
  - Threading
    - ExecutorService and Executors
    - Use thread to sort positive array
  - Sort negative array
  - Wait for positive sort to finish
  - Combine sorted arrays
  - Return combined array



```java
private static String[] sort(String[] toSort) throws InterruptedException, ExecutionException {
    ArrayList<String> positiveArray = new ArrayList<String>();
    ArrayList<String> negativeArray = new ArrayList<String>();

    for(String s: toSort) {
        if(s.contains(s: "-")) {
            negativeArray.add(s);
        }
        else {
            positiveArray.add(s);
        }
    }

    //Create thread and sort positive array
    ExecutorService pool =  Executors.newFixedThreadPool(nThreads: 1);
    final String[] pos =  positiveArray.toArray(new String[positiveArray.size()]);
    final int posLength = pos.length;
    Future<String[]> positiveSortedArray = pool.submit(()->{
      return KevinArhelgerSort.positiveBucketSort(pos, posLength);
    });

    //Sort negative array
    String[] neg =  negativeArray.toArray(new String[negativeArray.size()]);
    final int negLength = neg.length;
    KevinArhelgerSort.negativeBucketSort(neg, negLength);

    //Wait for positve sort to finish
    while(!positiveSortedArray.isDone()) {

    }

    //Shutdown executor pool
     pool.shutdown();

    toSort = Arrays.copyOf(neg, negLength+posLength);
    System.arraycopy(pos,srcPos: 0,toSort,negLength,posLength);

    return toSort;
}
```

Description of algorithm

- For each array (positive/negative), create arrays for buckets
  - Buckets assume fairly uniform distribution of numbers
  - Becket values range from 1 to the number of items in the array
- Sort each number into the correct bucket
  - bucket# = #items * number / 5
- Sort the buckets using insertion sort
  - Base compare function used from Group 0
- Concatenate the buckets back into the array



```java
//Sort positive array
static String[] positiveBucketSort(String arr[], int n)
{
    if (n <= 0)
        return null;

    // Create n empty buckets
    @SuppressWarnings("unchecked")
    ArrayList<String>[] buckets = new ArrayList[n+1];

    for (int i = 0; i <= n; i++) {
        buckets[i] = new ArrayList<String>();
    }

    // Put array elements in different buckets
    for (int i = 0; i < n; i++) {
        if(arr[i].contains(s: "/")) {
            String[] saFrac = arr[i].split(regex: "/");
            Double value = Double.parseDouble(saFrac[0]) / Double.parseDouble(saFrac[1]);
            double index = value * n/5;
             buckets[(int)index].add(arr[i]);
        }
        else{
            double value = Double.parseDouble(arr[i]);
            double index = value * n/5;
            buckets[(int)index].add(arr[i]);
        }
    }

    // Sort individual buckets
    for (int i = 0; i <= n; i++) {
        insertionSort(buckets[i], new Group5.SortingCompetitionComparator());
    }

    // Concatenate all buckets into arr[]
    int index = 0;
    for (int i = 0; i <= n; i++) {
        for (int j = 0; j < buckets[i].size(); j++) {
            arr[index++] = buckets[i].get(j);
        }
    }
    return arr;
}
```

Worst Case and Best case running times

Best Case: Θ(n)

- One item in each bucket

Average Case: Θ(n)

- Very Few items in each bucket

Worst Case: Θ(n^2)

- All items in one bucket, and sorted in descending order

Questions?

no