RADIX SORT

It takes a list of numbers, sorts them into buckets based of the digit in a specif index in the number. The iteration is repeated from the least significant digits to the most significant ones.

PSEUDOCODE

```
numberMap = new TreeMap        → create the container for the buckets
bucket # = new ArrayList       → create the buckets. Repeat it for the number
(...)                            from 0 to 9.
maxLen = 0;                    → will hold the length of the variable that is being compared
count = 1;                     → tracks the number of iterations of the main loop.
for each arrayItem {
    if value.length > maxLen {         → checks for the length of the largest value
        maxLen = value.length; }}
index = maxLen - 1             → position of the least significant digit
while (count < maxLen) {       → iterate as many times as the maximum number of digits
  for each bucket in the numberMap {
        clean bucket; }}               → clean all buckets before each iteration
  for each arrayItem {
      string value = numeric value to string;   → store the value as string
      char digit = 0;                  → default value to be added for issing digits
  adjustedIndex = index - (maxLen - value.length);             } the index has to be reajusted to support
  adjustedIndex = getAdjustedIndex (value, adjustedIndex);     } values that contain "." and "E".
                                                                 An outside function will be responsible for it
  char digit = value [adjustedIndex];   → if the adjusted index is valid, we store the digit
  switch case (digit) {
      case "#":                        → place the element into a bucket from one to nine,
          add to bucket #;               depending on the digit.
             (...) }
  int reAdd = 0;
  for each list in numbersMap {
       for each item in each list {    → we will go through the buckets and point each index
            originalArray [reAdd++] = arrayItem; }}   from the original array to that element inside the
                                         bucket in order from the larger values to smaller ones.
  count ++;
  index ++; }
}
```

```
numberMap = new TreeMap
bucket # = new ArrayList
(...)

maxLen = 0;
count = 1;
```

assignments = 22
comparisons = 0
arithmetic operations = 0

```
for each arrayItem {
        if value.length > maxLen {
                maxLen = value.length; }}
```

assignments = 2n
comparisons = n
arithmetic operations = 0

```
index = maxLen - 1
```

assignments = 1
arithmetic operations = 1

```
while (count < maxLen){
```

maxLen - 1 == n

```
   for each bucket in the numberMap {
          clean bucket; }}
```

assignments = 10n

```
   for each arrayItem {
        string value = numeric value to string;
        char digit = 0;
```

assignments = 2n

```
   adjustedIndex = index - (maxLen - value.length);
   adjustedIndex = getAdjustedIndex* (value, adjustedIndex);
   char digit = value [adjustedIndex];

   switch case (digit) {
        case "#":
              add to bucket #;
                 (...) }
```

assignments = 5n
comparisons = 14n
arithmetic operations = 4n

```
   int reAdd = 0;
   for each list in numbersMap {
          for each item in each list {
                originalArray [reAdd++] = arrayItem; }}

   count ++;
   index ++; }
}
```

assignments = n
comparisons = 0
arithmetic operations = n

ASSIGNMENTS = 23 + 20n
COMPARISONS = 15n
ARITHMETIC OPERATIONS = 1 + 5n

→ 40n + 24 → BIG O: O(n)