**3.**

Radix Sort

1. Find the max word length (== k) and go through the list, appending “0s” to the rest of the words in order to meet that length **[O(n)]**
2. Go through the list of words, putting each in one of 26 locations according to the numerical value of the last letter **[O(n)]**
3. Reorganize the list according to each word’s location **[O(n)]**
4. Repeat k (constant) number of times, evaluating one letter closer to the start each time

**[\* O(1)]**

Run time = O(1) \* [O(n) + O(n) + O(n)] → **O(n)**

**5.**

1. Go through the list and move each rational number to a new array (A1) **[O(n)]**
2. Find the max decimal length and append 0s to all other rationals in order to meet that length **[O(n)]**
3. Sort A1 using radix sort **[O(n)]**
4. Find the max natural number length and add 0s to the beginning of all other natural numbers in order to meet that length **[O(n)]**
5. Sort A using radix sort **[O(n)]**
6. Combine arrays into a new array A2 **[O(n)]**
7. If needed, erase contents of A and transfer A2 to A **[O(n)]**

Run time = O(n) \* 7 → **O(n)**

**6.**

1. Use bucket sort to sort the list of ranges according to the ai values **[theta(n)]**
2. Go through the sorted list, checking if each bi is less than ai+1 **[theta(n)]**

Run time = theta(n) \* 2 → **theta(n)**

**7.**

**i.** O(n): We go through the list once to verify that it’s sorted

**ii.** O((n + 1)!): Assuming that the random sequences can’t repeat, it will take n! times to reach a sorted list. We also need to check that the list is sorted every time, which means that the total time is n! \* n, or O((n + 1)!)

**iii.** No. The list is randomized, which means that it doesn’t take the original order into account. Therefore it won’t be stable.