CS 61B Fall 2025

Sorting III

Discussion 12: November 24, 2025

1 Radix Sorts

a) Sort the following list using LSD Radix Sort with counting sort. Show the steps taken after each round of counting sort. The first row is the original list and the last two rounds are already filled for you.

	30395	30326	43092	30315
1				
2				
3				
4	30315	30326	30395	43092
5	30315	30326	30395	43092

Solution:

The underlined sections denote the digits that have already been sorted.

	30395	30326	43092	30315
1	43092	30395	30315	30326
2	30315	30326	43092	30395
3	43092	30315	30326	30395
4	30315	30326	30395	43092
5	30315	30326	30395	43092

b) Sort the following list using MSD Radix Sort with counting sort. Show the steps taken after each round of counting sort. The first row is the original list and the first round is already filled for you.

	21295	22316	30753	21248	30751
1	21295	22316	21248	30753	30751
2					
3					
4					
5					

Solution:

	21295	22316	30753	21248	30751
1	21295	22316	21248	30753	30751
2	21295	21248	22316	30753	30751
3	21295	21248	22316	30753	30751
4	21248	21295	22316	30753	30751
5	21248	21295	22316	30751	30753

c) Give the best case runtime, worst case runtime, and stability for both LSD and MSD radix sort. Assume we have N elements, a radix R, and a maximum number of digits in an element W.

	Time Complexity (Best)	Time Complexity (Worst)	Stability
LSD Radix Sort			
MSD Radix Sort			

Solution:

	Time Complexity (Best)	Time Complexity (Worst)	Stability
LSD Radix Sort	$\Theta(W(N+R))$	$\Theta(W(N+R))$	Yes
MSD Radix Sort	$\Theta(N+R)$	$\Theta(W(N+R))$	Yes

d) We saw in part (c) that radix sort has good runtime with respect to the number of elements in the list. Given this fact, can we say that radix sort is the best sort to use?

No. Though radix sort runs linear with respect to the number of elements in the list, the runtime also depends on the size of the radix R and the length of the longest "word" W (or the number of digits in a number). Additionally, it is not always possible to use radix sort, because not all objects can be split up into digits. However, comparison sorts can be used on *any* object that defines a `compareTo` method, and would work well with `compareTo` methods that are fast.

2 LSD Radix Sort

In this question, we are trying to sort a list of strings consisting of **only lowercase alphabets** using LSD radix sort. In order to perform LSD radix sort, we need to have a subroutine that sorts the strings based on a specific character index. We will use counting sort as the subroutine for LSD radix sort.

(a) Implement the method **stableSort** below. This method takes in **items** and an **index**. It sorts the strings in **items** by their character at the **index** index alphabetically. It is stable and should run in $O(N)$ time, where N is the number of strings in **items**.

```
/* Sorts the strings in `items` by their character at the `index` index alphabetically.
This should modify items instead of returning a copy of it. */
private static void stableSort(List<String> items, int index) {
    Queue<String>[] buckets = new Queue[26];
    for (int i = 0; i < 26; i++) {
        buckets[i] = new ArrayDeque<>();
    }
    for (String item : items) {

        char c = item.charAt(index);

        int idx = c - 'a';

        buckets[idx].add(item);
    }

    int counter = 0;

    for (String item : items) {

        while (!bucket.isEmpty()) {
            items.set(counter, bucket.poll());
            counter++;
        }
    }
}
```

(a) Now, using the **stableSort** method, implement the method **lsd** below. This method takes in a **List** of **Strings** and sorts them using LSD radix sort. It should run in $O(Nc \cdot M)$ time, where N is the number of strings in the list and M is the length of each string.

```
public static List<String> lsd(List<String> items) {
    int length = items.get(0).length();


    for (int i = length - 1; i >= 0; i--) {

        stableSort(items, i);
    }

    return items;
    }
}
```

3 MSD Radix Sort

Now, let's solve the same problem as the previous part, but using a different algorithm. Recursively implement the method **msd** below, which runs MSD radix sort on a **List** of **Strings** and returns a sorted **List** of **Strings**. For simplicity, assume that each string is of the same length, and all characters are lowercase alphabets. You may not need all of the lines below.

In lecture, recall that we used counting sort as the subroutine for MSD radix sort, but any stable sort works! For the subroutine here, you may use the **stableSort** method from the previous question, which sorts the given list of strings in place, comparing two strings by the given index. Finally, you may find following methods of the **List** class helpful:

(a) **List<E> subList(int fromIndex, int toIndex)**. Returns the portion of this list between the specified **fromIndex**, inclusive, and **toIndex**, exclusive.

(b) **addAll(Collection<? extends E> c)**. Appends all of the elements in the specified collection to the end of this list, in the order that they are returned by the specified collection's iterator.

```
public static List<String> msd(List<String> items) {
    return msd(items, 0);
}

private static List<String> msd(List<String> items, int index) {
    if (items.size() <= 1 || index >= items.get(0).length()) {
        return items;
    }

    List<String> answer = new ArrayList<>();
    int start = 0;


    stableSort(items, index);
    for (int end = 1; end <= items.size(); end += 1) {
        if
(end == items.size() || items.get(start).charAt(index) != items.get(end).charAt(index)) {

            List<String> subList = items.subList(start, end);;

            answer.addAll(msd(subList, index + 1));

            start = end;
        }
    }
    return answer;
}

/* Sorts the strings in `items` by their character at the `index` index alphabetically. */
private static void stableSort(List<String> items, int index) {
    // Implementation not shown
}
```