CS 61B Fall 2025

Sorting III

Exam-Level Discussion 12: November 24, 2025

1 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 = _____;

        int idx = _____;

        _____;
    }

    int counter = 0;

    for (_____) {

        while (_____) {
            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 (_____) {

        _____;
    }

    return items;
    }
}
```

2 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 ____________________________________________;
}

private static List<String> msd(List<String> items, int index) {

    if (_____________________________________________) {
        return items;
    }

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


    ____________________________________________;
    for (int end = 1; end <= items.size(); end += 1) {

        if (_____________________________________________________________________) {

            ____________________________________________________________;

            ____________________________________________________________;

            ____________________________________________________________;
        }
    }
    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
}
```

3 Zero One Two Step

a) Given an array that only contains 0's, 1's and 2's, write an algorithm to sort it in linear time without creating a new array. You may want to use the provided helper method, swap. Hint: Consider how Hoare partitioning rearranges elements in an array.

```
public static void specialSort(int[] arr) {
  int front = 0;
  int back = arr.length - 1;
  int curr = 0;

  while (_____________________________) {
      if (arr[curr] < 1) {

        ____________________________________________________;

        ____________________________________________________;

        ____________________________________________________;
      } else if (arr[curr] > 1) {

        ____________________________________________________;

        ____________________________________________________;
      } else {

        ____________________________________________________;
      }
    }
  }
}

private static void swap(int[] arr, int i, int j) {
  int temp = arr[i];
  arr[i] = arr[j];
  arr[j] = temp;
}
```

b) We just wrote a linear time sort, how cool! Why can't we always use this sort, even though it has better runtime than Mergesort or Quicksort?

c) The sort we wrote above is also "in place". What does it mean to sort "in place", and why would we want this?