CS 61B — Sorting II

Spring 2025 — Exam-Level 12: April 21, 2025

1 Sorted Runtimes

We want to sort an array of N **unique** numbers in ascending order. Determine the best case and worst case runtimes of the following sorts:

see the solution !!!

(a) Once the runs in merge sort are of size $<= \frac{N}{100}$, we perform insertion sort on them.

Best Case: $\Theta(N)$, Worst Case: $\Theta(N\log N)$ N^2

(b) We use a linear time median finding algorithm to select the pivot in quicksort.

Best Case: $\Theta(\cancel{N^2})$ $N\log N$, Worst Case: $\Theta(N\log N$

(c) We implement heapsort with a min-heap instead of a max-heap. You may modify heapsort but must maintain constant space complexity.

Best Case: $\Theta(N\log N)$, Worst Case: $\Theta(N\log N$ $N\log N + N$

(d) We use any algorithm to sort the array knowing that:

- There are at most N inversions.

  Best Case: $\Theta(\cancel{N\log N})$ N, Worst Case: $\Theta(\cancel{N^2})$ N
- There is exactly 1 inversion.

  Best Case: $\Theta(\cancel{N})$ 1, Worst Case: $\Theta(\cancel{N^2})$ N
- There are exactly $\frac{N(N-1)}{2}$ inversions.

  Best Case: $\Theta(\cancel{N\log N})$ N, Worst Case: $\Theta(\cancel{N^2})$ N

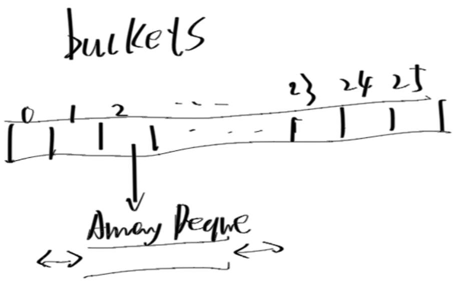
$N + (N-1) + (N-2) + \cdots + 1$

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].offer(item);
    }
    int counter = 0;

    for (int i=0 ; i<26 ; i++) {

        while (buckets[i].peek() != null) {

            items.set(counter, bucket.poll());

            counter++;
        }
    }
}
```



Refer to standard solution,

(b) 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 $O(N \cdot M)$ 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 index= length-1 ; index>=0 ; index--) {

        stableSort (items, index);
    }

    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:

1. `List<E> subList(int fromIndex, int toIndex)`. Returns the portion of this list between the specified `fromIndex`, inclusive, and `toIndex`, exclusive. [,)
2. `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;
    stable(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
}
```