CS 61B Spring 2024

More Sorting

Exam-Level 12: April 15, 2024

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:

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

Best Case: $\Theta($ $)$, Worst Case: $\Theta($ $)$

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

Best Case: $\Theta($ $)$, Worst Case: $\Theta($ $)$

(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: $\Theta($ $)$, Worst Case: $\Theta($ $)$

(d) We run an optimal sorting algorithm of our choosing knowing:

- There are at most N inversions.

  Best Case: $\Theta($ $)$, Worst Case: $\Theta($ $)$

- There is exactly 1 inversion.

  Best Case: $\Theta($ $)$, Worst Case: $\Theta($ $)$

- There are exactly $\frac{N(N-1)}{2}$ inversions

  Best Case: $\Theta($ $)$, Worst Case: $\Theta($ $)$

2 MSD Radix Sort

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. 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, 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 ____________________________________________________________;
}

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 Shuffled Exams

For this problem, we will be working with `Exam` and `Student` objects, both of which have only one attribute: `sid`, which is a integer like any student ID.

PrairieLearn thought it was ready for the final. It had meticulously created two arrays, one of `Exams` and the other of `Students`, and ordered both on `sid` such that the ith `Exam` in the `Exams` array has the same `sid` as the ith `Student` in the `Students` array. Note the arrays are not necessarily sorted by `sid`. However, PrairieLearn crashed, and the `Students` array was shuffled, but the `Exams` array somehow remained untouched.

Time is precious, so you must design a $O(N)$ time algorithm to reorder the Students array appropriately **without** changing the `Exams` array!

Hint: While you cannot modify the `Exams` array, you can sort a copy of the `Exams` array with some added information. Think about what information would be useful to put back the `Students` array in the same order as the exams.

.