CS2040S: Data Structures and Algorithms

Exercises/Optional Problems for Week 2

For: January 23, 2020

Problem 1. Inversion Madness

Recall the definition of an inversion as stated below:

Definition 1 *Given a list of elements, and a partial order $\prec$ on the elements, the pair (a, b) forms an **inversion** in the list if $a \prec b$ but b appears before a in the current list order.*

Intuitvely, the number of inversions tells us the number of **consecutive** swaps required in order to take a list from its unsorted order into its sorted order. (Remember this is different from the minimum number of swaps, which is achieved by selection sort!)

Now counting inversions are particularly useful for a whole lot of purposes. Operating systems can use them to determine the amount of work required to reorganise the disk in some order. In a more unusual setting, it can be used to determine the distance of a student's answer of sorting recurrences from the actual answer which helps tutors determine how many marks to deduct :D.

In any sense, you are given the class `InversionCounter.java` in the zip file, which should look something like this:

```
    class InversionCounter {

    public static long countSwaps(int[] arr) {
        return 0;
    }

    public static long mergeAndCount(int[] arr, int left1, int right1, int left2, int right2) {
        return 0;
    }
}
```

There are two functions present in this class which are:

- `long countSwaps(int[] arr)` Given an input array, this counts the number of swaps required to sort the array (i.e. the number of inversions)
- `long mergeAndCount(int[] arr, int left1, int right1, int left2, int right2)` Given an input array so that arr[left1] to arr[right1] is sorted and arr[left2] to arr[right2] is sorted (also left2 = right1 + 1), merges the two so that arr[left1] to arr[right2] is sorted, and returns the minimum amount of adjacent swaps needed to do so.

IMPORTANT NOTE: Notice that the inversion counts can be quite huge. Hence, notice that method signatures for both `countSwaps` and `mergeAndCount` are of the `long` type (which represents 64 bit integers in Java). Please ensure that whenever you do your counting you handle them in a proper manner such that no overflow errors occur, especially when using `int` instead of `long`.

You may use the given sample tests given in `InversionCounter.java` to verify the correctness of your program and run it against sample tests. You are encouraged to write your own tests as well. When ready for submission, submit only the file `InversionCounter.java` onto Coursemology.