Homework Sheet 3

Authors	**Tutors**
Abdullah Oğuz Topçuoğlu	Maryna Dernovaia
Ahmed Waleed Ahmed Badawy Shora	Jan-Hendrik Gindorf
Yousef Mostafa Farouk Farag	Thorben Johr

Exercise 3

We can do something similar to LsdRadix sort we saw in the lecture.
Pseudocode:

```
function SortGridPoints(A[1..n]) {
    redefine key(point) := point.y
    CountingSort(A)

    redefine key(point) := point.x
    CountingSort(A)
}
```

Correctness:

- The first CountingSort sorts the points by their y coordinates.
- The second CountingSort sorts the points by their x coordinates, but since CountingSort is stable, the order of points with the same x coordinate is preserved.
- Therefore after both sorts the points are sorted lexicographically by (x, y).
- Thats also what we did in the lecture for LSDRadixSort we started from the least significant digit to the most significant digit. The same idea applies here.

Running Time Analysis:

- Each CountingSort runs in time O(n + k) where k is the range of the keys.
- Here the keys are the x and y coordinates of the points.
- Since the points are connected, the range of x coordinates is at most n and the range of y coordinates is also at most n.
- Therefore each CountingSort runs in time O(n + n) = O(n).
- Since we perform two CountingSorts, the total running time is O(n) + O(n) = O(n).

Exercise 4

(a)

We can use the LSDRadixSort again.
Pseudocode:

```
const ALPHABET_SIZE = 26
function SortStrings(A[1..n]) {
    for i in ALPHABET_SIZE..1 {
        redefine key(string) := string[i]
        CountingSort(A)
    }
}
```

Correctness:

- We sort the strings starting from the last character to the first character (least significant to most significant).
- Each CountingSort is stable, so the order of strings with the same character at position i is preserved.
- Therefore after sorting by all character positions, the strings are sorted lexicographically.
- So just like LSDRadixSort we saw in the lecture where d is alphabet size, U is the alphabet

Running Time Analysis:

- Each CountingSort runs in time O(n + k) where k is the range of the keys.
- Here the keys are the english letters 'a' to 'z', so k = 26.
- Therefore each CountingSort runs in time O(n + 26) = O(n).
- Since we perform CountingSort 26 times, the total running time is O(26 * n) = O(n).