# Report Assignment 2

Amelie Löwe, Håkan Johansson

October 18, 2019

## Exercise 2

We used arrays to keep track of the coordinates we already visited in the grid. Therefore we have a variable for the current coordinate and by comparing the visited ones with the current, we can see when an intersection occurs.

### getIntersection()

The time complexity of the *getIntersection()* method is $O(N)$. Since we loop through the array two times, the time complexity would be:

$$O(2N) = O(N).$$

### Constructor

The only function of the constructor in the MyItinerary class is to initialize the fields.

## Exercise 3

### isSameCollection()

First of all we compare the lengths of the arrays. If the length is not equal, we immediately return false. Otherwise we loop through both of the arrays to check if they have the same integer values.

#### Time Complexity Analysis

Worst case in our solution is time complexity

$$O(N^2),$$

and the best case is when the lengths of the arrays are different. Then we have constant time complexity

$$O(1).$$

### minDifferences()

Similarly to the isSameCollection() method, the length of the arrays gets compaired. If the length of the arrays are different, we throw an exception. Furthermore we sort the arrays using the private insertionSort() method. Then we loop through the arrays and sum the squared distances.

### Time Complexity Analysis

Since we are sorting both arrays using the insertionSort() method, the time complexity is $2O(N^2)$. Because we also loop through the arrays once after sorting them, we will get follwing expression

$$T(N) = O(N^2) + O(N^2) + O(N) = 2O(N^2) + O(N) = O(N^2).$$

## getPercentileRange()

First we need to check if upper limit is less than lower limit. If that is the case, we throw an exception. After that we need to consider a few different things. The first thing to calculate is how many percent each element is. That is because we want to take that number and divide with the difference between upper and lower, to see how big the new array needs to be.

Next we sort the array so it will be easier to just loop through and append the numbers in the given positions to the new array.

### Time Complexity Analysis

Since we are sorting the arrays using the insertion sort method, we have time complexity $O(N^2)$. Furthermore we loop through the array once, which gives us linear time complexity $O(N)$. The entire expression will be

$$T(N) = O(N^2) + O(N) = O(N^2).$$