Frederick Wittman

Lars Kotthoff

Rajiv Khadka

COSC 3020-01

Lab 04

10/04/19

I implemented the algorithm with three functions.

The first function generates all the combinations of a string of numbers. The time complexity of this function cannot exceed the number of ways we can permute a string of numbers. The number of ways we can permute a string of numbers is the number of arrangements we can make with them. This is well known to n!, and similarly the time complexity of the first function is O(n!) + c = O(n!).

The second function has constant time. It calls the first function to generate the list of combinations and then calls an is-sorted function on each element of the list. For a string of numbers of length n, the is-sorted function is ϴ(n).

Putting it all together, the time complexity of the algorithm is therefore given by O(n!) * ϴ(n) = O(n! * n). In the best case, the algorithm generates the set of combinations and then finds the sorted string right away. The time complexity here is therefore O(n! + n) = O(n!). In the worst case, the sorted string is found at the end of the set of combinations. The time complexity in this case is given by O(n! * n).

If permutations were generated randomly rather than systematically, the best case would occur when the first random generation is sorted. The time complexity here is just the time it takes to verify that the list is sorted: ϴ(n + c) = ϴ(n). In the worst case, for n > 1, the sorted permutation is never found, giving time complexity O(∞).

Note: I am aware of the distinction between combination and permutation. Since I borrowed the permutation function, I thought this analysis would be more appropriate.