Common Runtimes of Algorithms

This document contains summary of some of the items discussed as part of the COMS 3110 Lectures. This document **does not replace the lecture materials**. This document may contain some topics that are not covered as part of the lecture; you will not be tested on those parts, they are made available to you for gaining further knowledge on topics/concepts that are related to class-lecture.

We typically evaluate efficiency of an algorithm using asymptotic bounds, which describe how the (worst-case or expected) runtime grows as the input size n becomes large. These bounds are expressed using Big-O notation, which captures the growth rate for large n while ignoring constant factors and lower-order terms.

- **constant time:** $O(1)$, e.g., accessing an element in an array by index.
- **logarithmic time:** $O(\log n)$, e.g., binary search on a sorted array.
- **linear time:** $O(n)$, e.g., finding the maximum element in an unsorted array, merging two sorted array
- **polynomial time:** $O(n^k)$ for some constant $k \geq 1$, e.g., selection sort
- **exponential time:** $O(2^n)$, $O(n2^n)$, etc.

Remark: While asymptotic bounds give some insight on the scalability of an algorithm and are useful for comparing efficiency of different algorithms, the actual performance also depends on many factors. You learn these theoretical tools to build a solid foundation to help identify which parts of the code are worth optimizing. But don't forget the practical side. For example, if you expect the input side to be small, focusing too much on improving the asymptotic complexity may not lead to meaningful performance gains.