Module 1 - Material Notes

Algorithmic Exercises

https://projecteuler.net/archives (https://projecteuler.net/archives)

https://www.hackerrank.com/ (https://www.hackerrank.com/)

Big O Notation

Big O notation is a mathematical notation used in computer science to describe the performance or complexity of an algorithm. Specifically, it expresses the upper bound of the time complexity in the worst-case scenario, based on the size of the input.

Here are some key terms and concepts related to Big O notation:

Time Complexity: This is a measure of the amount of time an algorithm takes to run, as a function of the size of the input to the program. It's generally expressed using Big O notation.

Space Complexity: This is a measure of the amount of memory an algorithm needs to run, also expressed as a function of the size of the input. It can also be represented in terms of Big O notation.

Worst Case: The worst-case scenario for an algorithm is the most unfavorable situation for it, where it performs the maximum number of operations. In Big O notation, we express the worst-case time complexity.

Asymptotic Behavior: Big O notation describes the limiting behavior of a function when the argument tends towards a particular value or infinity, usually in terms of simpler functions.

Here are some common Big O notations, from fastest to slowest growth:

1. **O(1)**: Constant time complexity. The algorithm takes the same amount of time to complete, regardless of the input size. Example: looking up a single element in an array.
2. **O(log n)**: Logarithmic time complexity. The running time increases logarithmically with the size of the input. Example: binary search algorithm.
3. **O(n)**: Linear time complexity. The running time increases linearly with the size of the input. Example: a single loop over all elements of an array.
4. **O(n log n)**: Log-linear time complexity. This is better than quadratic time but worse than linear time. Example: efficient sorting algorithms like quicksort and mergesort.
5. **O(n^2)**: Quadratic time complexity. The running time is proportional to the square of the size of the input. Example: simple sorting algorithms like bubble sort and selection sort.
6. **O(n^3)**: Cubic time complexity. The running time is proportional to the cube of the size of the input. Example: matrix multiplication.
7. **O(2^n)**: Exponential time complexity. The running time doubles with each addition to the input size. Example: the naive recursive Fibonacci sequence algorithm.
8. **O(n!)**: Factorial time complexity. The running time grows factorialy with the size of the input. Example: solving the traveling salesperson problem via brute-force search.