## Algorithms & Data Structures I Week 12 Lecture Note

| | | | |
|---|---|---|---|
| **Notebook:** | Algorithms & Data Structures I | | |
| **Created:** | 2020-10-21 4:14 PM | **Updated:** | 2021-01-16 4:10 PM |
| **Author:** | SUKHJIT MANN | | |

| **Cornell Notes** | **Topic:** What makes a good algorithm? Part 2 | Course: BSc Computer Science |
|---|---|---|
| | | Class: CM1035 Algorithms & Data Structures I [Lecture] |
| | | Date: January 16, 2021 |

**Essential Question:**

What makes a good algorithm?

**Questions/Cues:**

- What is the time complexity?
- What is the space complexity of an algorithm?
- What is worst-case analysis or worst-case input?

Notes

- Time complexity = the quantification of the number of time-steps required to carry out an algorithm. Complexity just means the resources required.
- Space complexity (Algorithm) = is the amount of memory or space required to carry out the algorithm. We represent the complexity as a Big O class, but this will be the smallest Big O class in which the number of operations or space requirements live
- Worst case analysis or input = is the input that results in the most operations, or the input with the largest time complexity
    - Once we establish what the worst-case is, to find the worst-case time complexity, we establish the smallest Big O class in which the number of operations as a function lives
        - Worst case for linear search is O(n)
        - Best case for Bubble sort is O(n) and worst-case time complexity is O(n^2)
        - Worst-case time complexity for Insertion sort is O(n^2) like if the input array/vector is already sorted in reverse order and the best case is O(n) when the input is a almost-sorted array or vector
    - Since a digital computer can store a byte in each register, it can be assumed that the input can be an array or a string of bits with the possibility of storing nothing
    - A number n can be stored using O(log n) bits
    - So we can now have a problem where our number n is stored in an array of length O(log n), we call this the size of our input. For simplicity, let's refer to the size of our input as m.
        - Refer back to our algorithm for computing the factorial of n, it had the time complexity of O(n), but now if want to know the time complexity in terms of the size m, it will be $O(2^m)$ since the number n is exponentially larger than the size m. That is, for the amount of space our computer uses to store a number, the number of time-steps or operations required to compute the factorial on that number grows exponentially in that space or size of the input

**Note** When comparing the performance of algorithms, the key thing to remember is that the smaller the Big O class associated with the worst-case time complexity is, the better the algorithm performs. In other words, the smaller the class, the fewer time resources we need to implement an algorithm in the worst case

## Summary

In this week, we learned about worst-case time complexity and the size of an input