Unit 3 – Algorithms

Time Complexity & Big O Notation

1. Time complexity of algorithms is part and parcel of a deep understanding of JS and software engineering in general.
2. How do we assess time complexity in algorithms?
3. An algorithm is…
   1. A step by step set of instructions.
   2. Most of the “business logic” (determining how data can be stored/created/changed) is algorithmic.
      1. Image processing
      2. Recommendations
      3. File compressions
      4. Targeting advertising
4. The efficiency of an algorithm is about how much time an algorithm takes depending on the size of the input.
   1. If the time it takes does not depend on the size of the input, it is **constant**
      1. O(1)
      2. O => “Order of growth”
      3. And the input (n) is the input size.
   2. **It is the relationship between the size of an input and the number of computational steps.**
      1. Constant -> O(1)
      2. Linear -> O(n) (proportional growth of computational steps)
      3. Logarithmic O(logn)
      4. Quadratic O(n2)
      5. Exponential
5. Space Complexity
   1. As the input size grows, how much more “space” do we need?
      1. In memory
      2. If the variables inside of a “for” loop get garbage collected, that does not count towards space complexity.
6. The balance between time and space complexity:
   1. Usually, there is a tradeoff between time and space;
      1. As we improve time complexity, space complexity will usually increase.
      2. And vice versa.
      3. But if we had to prioritize one or the other, **time complexity** is the goal
      4. **Time complexity** matters because this impacts user retention and bounce rates.
      5. But additional memory/space can always be purchased.
7. Beyond:
   1. Don’t just try to memorize code shapes;
      1. For loops and nested loops don’t automatically determine the time complexity.