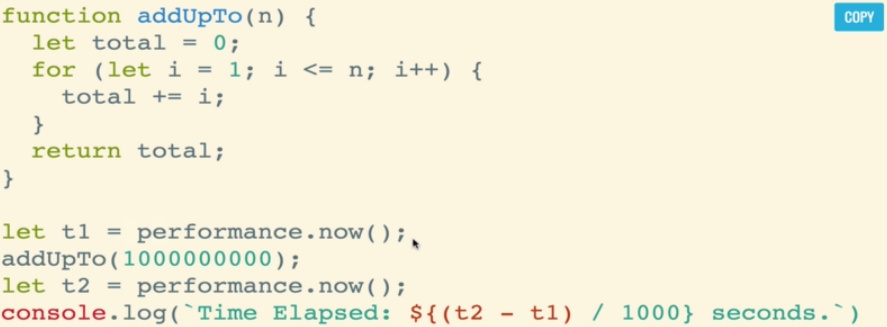
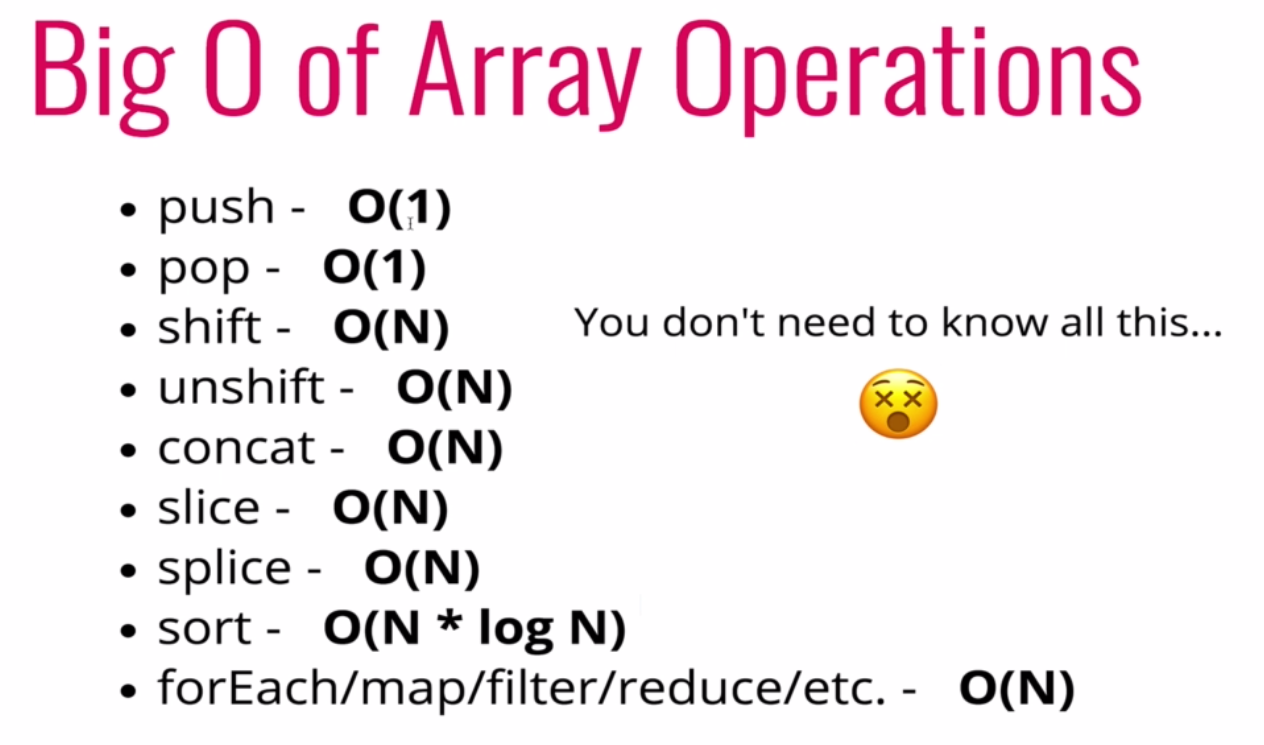
1. Big O
   1. Timing our code
      1. We can time our code by doing this:



* + 1. This will produce different time on the same machine and different times on different machines
    2. This is not efficient and we should not have to run a program to tell how fast it is
  1. Counting Operations
     1. We can count add, subtract, multiplication, division, and etc
     2. Thiss may prude something like 5n + 2 but we don’t care. Only the trend matters
  2. Big O rules
     1. Constants don’t matter, smaller terms don’t matter
     2. O(2n) -> o(n), O(500) -> O(1), O(13n^2) -> O(n^2), etc
     3. Big O short hands
        1. Arithmetic operations are constants
        2. Variable assignment is constant
        3. Accessing elements in an array (by index) or object (by key) is constant
        4. In a loop, the time complexity is the length of the loop times the complexity of whatever happens inside of the loop

1. Space complexity
   1. Auxiliary space complexity refers to space required by the algorithm, not including space taken up by the inputs
   2. Most primitives (Booleans, numbers, undefined, null) are constant space
   3. Strings are O(n) space where n is the string length
   4. Reference types are generally O(n), where n is the length (for arrays) or the number of keys (for objects)
2. Big O of arrays
   1. Insertion : depends
   2. Removal : depends
   3. Searching : O(n)
   4. Access O(1)
   5. Ex



1. Problem solving techniques
   1. An algorithm is a process or set of steps to accomplish a certain task
   2. Problem solving
      1. Understand the problem
      2. Explore concrete examples
      3. Break it down
      4. Solve and simplify
      5. Look back and refactor
   3. Understand the problem
      1. Can I restate the problem
      2. What are the inputs
      3. What are the outputs
      4. Can outputs be determined from the inputs (do you have enough info
      5. How should I label the important pieces of data that are part of the problem