Big O Notation

1. Describes the performance of an algorithm.
2. Describes if an algorithm is scalable or not?
   1. Meaning is an algorithm going to scale well as the input grows really large.
3. O(1)
   1. Algorithm runs in constant time regardless of input size.
4. O(n)
   1. As n (the size of the input) grows, the cost of the algorithm also grows linearly.
   2. If you have O(n + m), you can just call it O(n) because the cost of the algorithm grows linearly.
5. O(n^2)
   1. As input grows, the cost of the algorithm grows quadratically.
6. O(log n)
   1. Algorithm cost stops growing after a certain point, so it’s more efficient and scalable than O(n)
   2. Algorithms where the work is reduced by half in every step
   3. Ex: Binary Search
7. O(2^n)
   1. Logarithmic curve slows down as input size grows
   2. Exponential curve grows faster and faster as input size grows.
   3. Not scalable at all.
   4. Cost of algorithm grows very quickly as input size grows.
8. Space-Time Complexity
   1. How much memory an algorithm takes.

A screen shot of a computer program

Description automatically generated

* + 1. We have a loop variable (int i) that is independent of the size of the input.
    2. Whether our array has 10 or one million items, this method will only allocate some additional memory for this loop variable.
    3. Therefore, it takes O(1) space.

A screen shot of a computer program

Description automatically generated

* 1. Now we have a separate array that is equal in size to the input array.
     1. If our input array has a thousand items, this array will also have a thousand items.
     2. Therefore, space complexity is O(n)
  2. Input is not a factor when determining space complexity.
  3. Space complexity is the ADDITIONAL space that should be allocated relative to the size of the input.