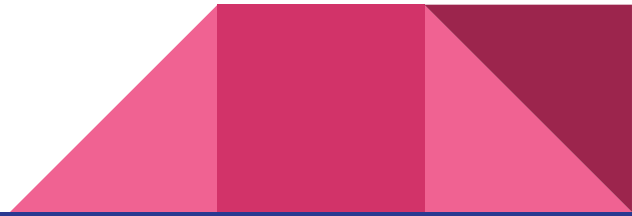
Big 0 Notation

Do Now

How can we compare the performance of different sorting algorithms?

Should we record the start time, run the algorithm, record the end time, and then calculate the running time?

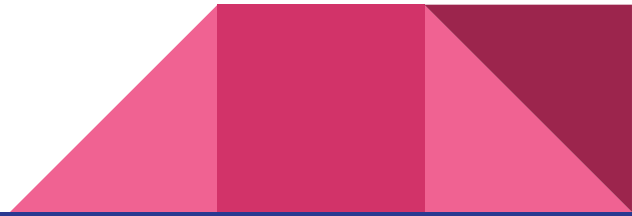


COMPLEXITY

There are two types of complexity:

- Time complexity: number of steps involved to run an algorithm.
- Memory complexity: amount of memory it takes to run an algorithm.

We should be concern about time complexity when developing our algorithms.



How do we measure the time complexity of an algorithm?

Looking at:

1. The worst case?
2. The best case?
3. The average case?

How do we measure the time complexity of an algorithm?

Looking a the best case does not help. You rarely going to have the best case.

Looking at the average case is not going to tell you the absolute worst time complexity.

Looking at the worst case can help us to understand what to expect

from the algorithm. That is why is more helpful to look at the worst case.

Example: Algorithm Add Sugar to Tea

1. Fetch the bowl containing the sugar
2. Get a spoon
3. Scoop out sugar using the spoon
4. Pour the sugar from the spoon into the tea
5. Repeat steps 3 and 4 until you have added the desired amount of sugar

Example: Algorithm Add Sugar to Tea

Number of sugars	Steps required
1	4
2	6
3	8
4	10

Time complexity depends on the number of sugars someone wants on their tea.

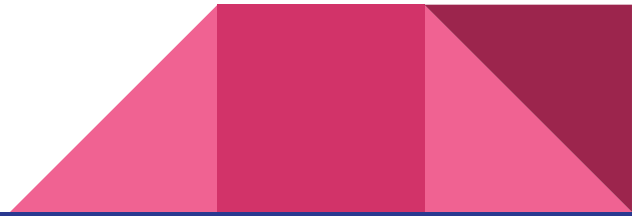
The Big 0 Notation

It is a way of expressing the complexity related to the number of items

that an algorithm has to deal with.

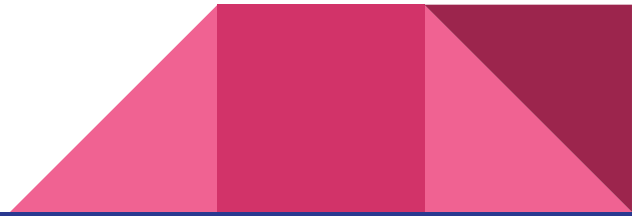
It is written as a capital O followed by an expression in parenthesis. Example: O(n)

It is conventional to designate the number of items by n.



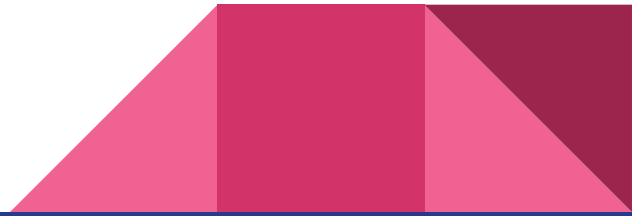
Big O notation for the number of sugars algorithm

- Number of desired sugars = n
- Total number of steps = 2n + 2 (as n grows the number of steps grows)
- The 2 in 2n and the +2 remain constant. They do not factor into the time complexity
- Time complexity = O(n)
- Linear time complexity

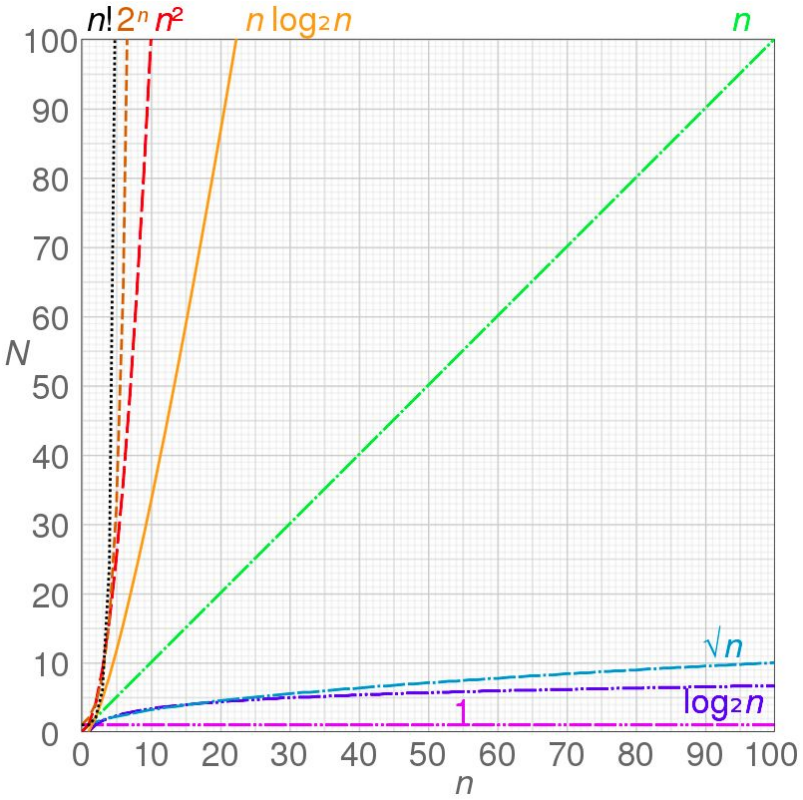


Big 0 values

Big 0 values	
O(1)	Constant
O(logn)	Logarithmic (base2)
O(n)	Linear
O(nlogn)	n logn
$O(n^2)$	Quadratic



Big O Notation



Graphs of functions commonly used in the analysis of algorithms, showing the number of operations *N* versus input size *n* for each function.

Source: Wikipedia