19CSE212 Lecture 0: Introduction/Refresher to Data Structures

by Ramkumar Narayanan

Contents

1 Recap of the seven growth functions

Asymptotic analysis represents growth rates of a function. The different functions we learnt are

1. ***Constant Function***: it does not matter what the value of n is; $f(n)$ will always be equal to the fixed constant value of c.

$$f(n) = c$$

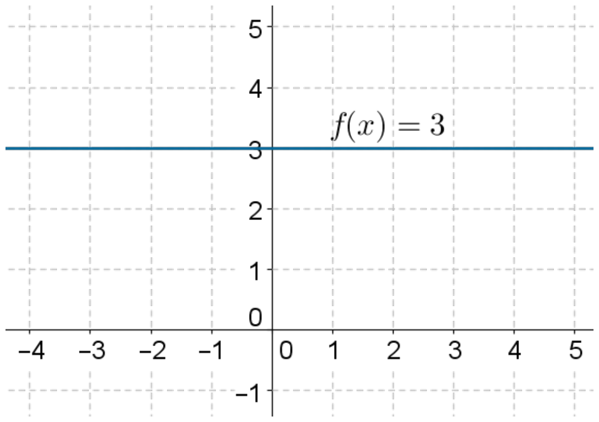


Figure 1: Constant function when $c = 3$

2. ***Logarithimic Function***: you will find this function ubiquitously while learning algorithms. The most common base for a logarithmic function is base 2.

$$f(n) = log_b n$$

b denotes the base.

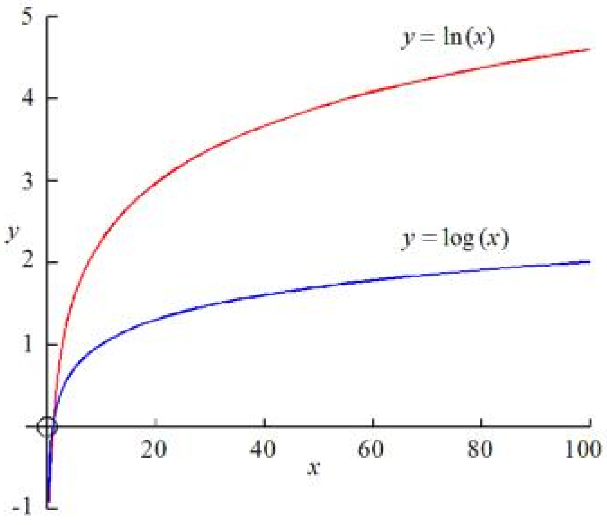


Figure 2: Logarithmic Function

3. ***Linear Function***: The function linearly rises with the value of n

$$f(n) = n$$

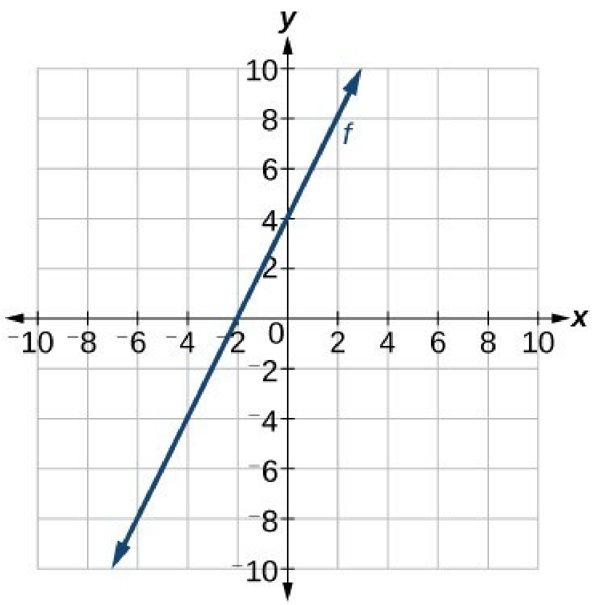


Figure 3: Linear growth rate

4. ***N-Log-N Function***: function assumes the growth rate is $n * log(n)$

$$f(n) = nlog(n)$$

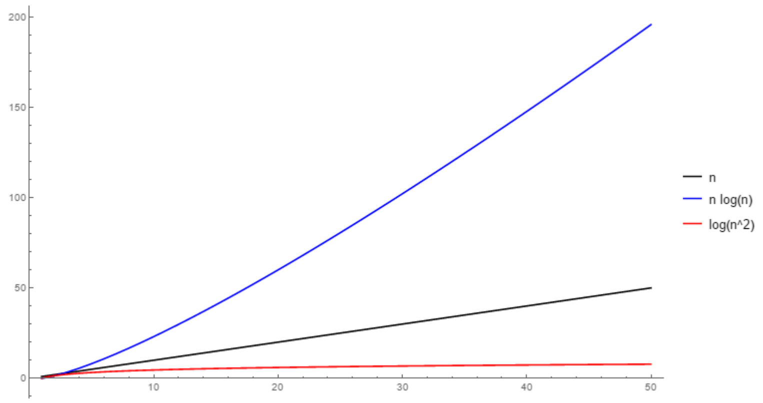


Figure 4: $nlog(n)$ growth

5. ***Quadratic Function***: function assumes the growth rate is n^2

$$f(n) = n^2$$

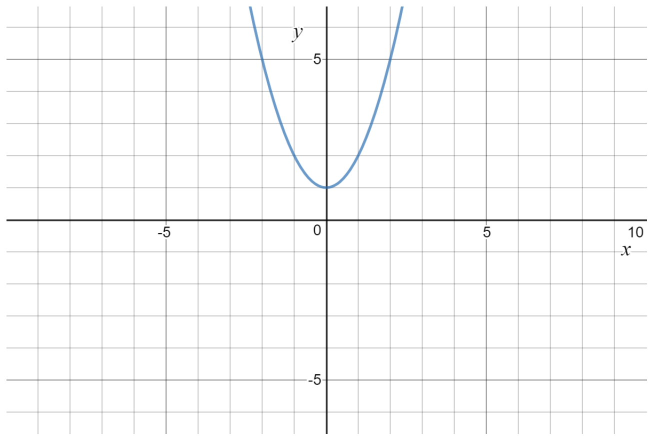


Figure 5: Function represents quadratic growth rate of $y = x^2 + 1$

6. ***Cubic Function***: function assumes the growth rate is n^3

$$f(n) = n^3$$

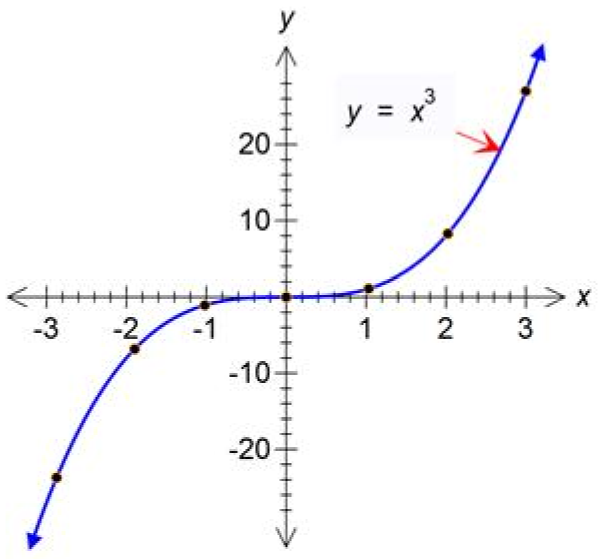


Figure 6: Function represents quadratic growth rate of $y = x^3$

7. ***Exponential Function***: function assumes the growth rate is b^n

$$f(n) = b^n$$

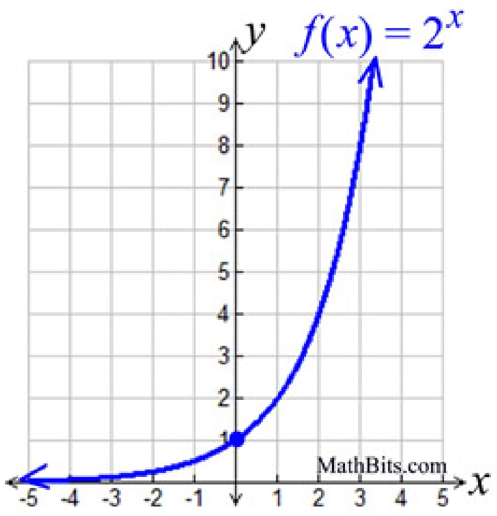
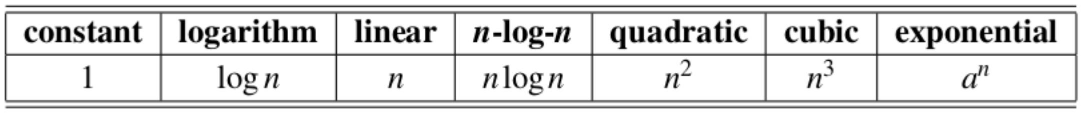


Figure 7: Function represents quadratic growth rate of $y = 2^x$

Each of them have varying growth rates. Some grow faster than the others. If we were to sort the functions based on the effect of growth (or) their growth rates we would get this.



constant	logarithm	linear	n-log-n	quadratic	cubic	exponential
1	$\log n$	n	$n \log n$	n^2	n^3	a^n

Figure 8: Sorted Functions from the least to most growth

2 Recap of Asymptotic Analysis

- The ***Big-Oh*** : ***O*** notation. It is often necessary to analyze the run time complexity or the speed at which an algorithm runs. We use the Big-Oh notation for that. Remember, we just want the rate of growth, we do not care about the lower order factors or constants. Below are some of the 'Big-Oh' notations of several functions.

  1. $f(n) = 5n^2 + 3nlogn + 2n + 5$ is $O(n^2)$
  2. $f(n) = 20n^3 + 10nlogn + 2n + 5$ is $O(n^3)$
  3. $f(n) = 3logn + 2$ is $O(logn)$
  4. $f(n) = 2^{n+2}$ is $O(2^n)$
  5. $f(n) = 2n + 100logn$ is $O(n)$

  The ***Big-Oh*** notations give us the upper limit of growth hence used to calculate the worst case performance.

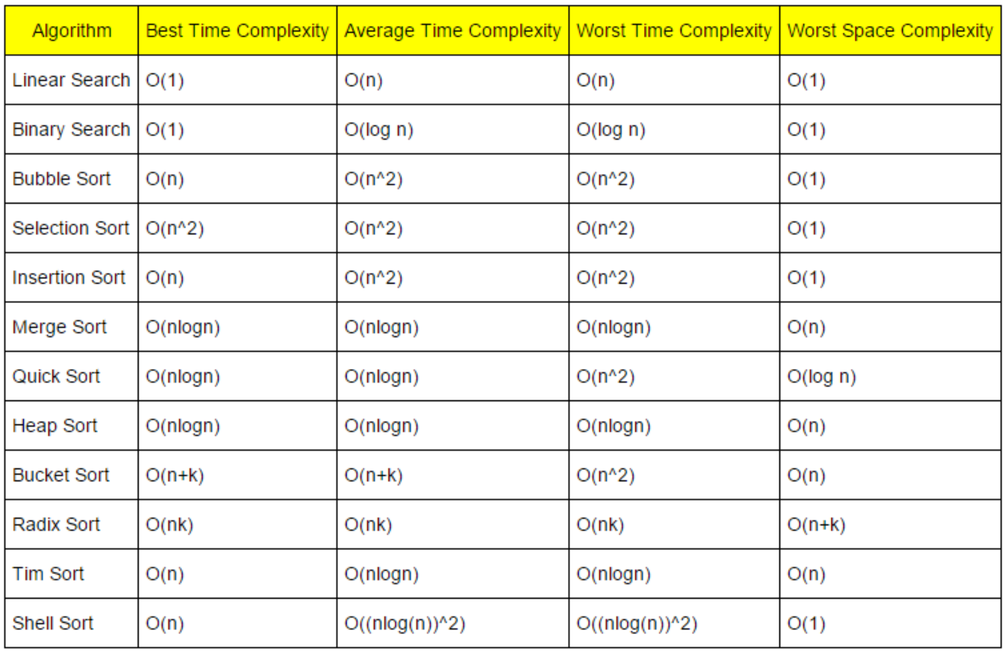
- The ***Big-Omega***: this notation provides "lesser than or equal to" notation of growth. Provides the asymptotic lower bound of the function; i.e. the best case runtime.

- The ***Big-Theta***: if we want precision upto a constant value, we use the Theta notation. In this case we don't want the upper bound or lower bound, we aim at obtaining the function that is the closest fit to the runtime.

Mostly, we will be using the Big-Oh notation for our analysis.

3 Recap of Searching and Sorting

Below chart gives a quick refresher on the various sorting and searching algorithms that you have learnt the introduction to Data Structures course. Be sure you know how asymptotic analysis is done for the common sort and search functions. For a quick refresher on how the sorting is done visit this page.



Algorithm	Best Time Complexity	Average Time Complexity	Worst Time Complexity	Worst Space Complexity
Linear Search	O(1)	O(n)	O(n)	O(1)
Binary Search	O(1)	O(log n)	O(log n)	O(1)
Bubble Sort	O(n)	O(n^2)	O(n^2)	O(1)
Selection Sort	O(n^2)	O(n^2)	O(n^2)	O(1)
Insertion Sort	O(n)	O(n^2)	O(n^2)	O(1)
Merge Sort	O(nlogn)	O(nlogn)	O(nlogn)	O(n)
Quick Sort	O(nlogn)	O(nlogn)	O(n^2)	O(log n)
Heap Sort	O(nlogn)	O(nlogn)	O(nlogn)	O(n)
Bucket Sort	O(n+k)	O(n+k)	O(n^2)	O(n)
Radix Sort	O(nk)	O(nk)	O(nk)	O(n+k)
Tim Sort	O(n)	O(nlogn)	O(nlogn)	O(n)
Shell Sort	O(n)	O((nlog(n))^2)	O((nlog(n))^2)	O(1)

Figure 9: Big-Omega, Big-Theta, Big-Oh, space complexity analysis of sorts and searches

4 Recap of Recursion

Recursion is a tricky topic to understand. Let us do some recap and practice before we get into the main topics of this course. Remember, recursion has two parts to it

1. ***base case***: This is the 'naive' case for which you know the answer to by heart.
2. ***general case***: This is the case where you take a chunk of the problem and off load the rest to the others.

Lets take a look at some examples

- ***Factorial***

```
def fact(n):
if(n==1): # base case
   return(1)
else:
   return n*fact(n-1)

print(fact(5))
```

- ***Fibonacci Series***

```
def fibo(n):
   if(n<=1): # base case
      return(n)
   else: # general case
      return fibo(n-1)+fibo(n-2)

print(fibo(3))
```

- ***Sum of Digits***

```
def sum_of_digits(n):
   if(n//10==0): # base case
      return(n)
   else: # general case
      return n%10+sum_of_digits(n//10)

print(sum_of_digits(123456789))
```