Basic Properties of Big-$\mathcal{O}$

Time complexity described the amount of computational time an algorithm takes. As this can vary and run times are fast for small n, we describe the behavior as n gets large for best, average, and worst case scenarios -mostly we focus on worst, i.e., Big-O. Here's a list of some functions listed from fast to slow[1]:

$\mathcal{O}(1)$ **Constant**

$\mathcal{O}(\log(n))$ **Logarithmic**.

Ignore Log bases: $\log_{10}(n) = \log(n)$

Same growth as $\log(n^k) = k\log(n)$

$\mathcal{O}(\log^c(n))$ **Polylogarithmic**

Don't forget that: $\log^c x = (\log x)^c$.

$\mathcal{O}(n)$ **Linear**

$\mathcal{O}(n\log(n))$ **Linearithmic/Loglinear**

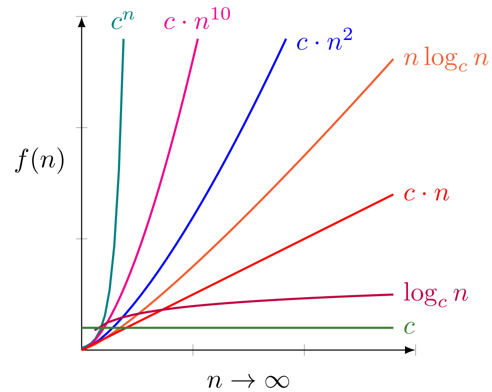
$\mathcal{O}(n^2)$ **Quadratic**

$\mathcal{O}(n^2\log(n))$

$\mathcal{O}(n^3)$ **Polynomial**

$\mathcal{O}(2^n)$ **Exponential**

$\mathcal{O}(n!)$ **Factorial**



Some properties
We ignore coefficients and log bases.

ex. $7x^2$ is $\mathcal{O}(x^2)$ and $3\log_{20} x$ is $\mathcal{O}(\log x)$ [2]

When adding functions, we only take the slowest function's Big-$\mathcal{O}$.

ex. $0.01x^3 + 12x^2 - 2\log 37x$ is $\mathcal{O}(n^3)$

When multiplying functions, we take take the product of the functions' Big-$\mathcal{O}$.

ex. $x^2+2 \in \mathcal{O}(n^2)$ and $\log_{10}(x) \in \mathcal{O}(\log n)$ so then $(x^2+2)(\log_{10}(x))$ is $\mathcal{O}(n^2\log(n))$

Sort function	Average/Typical	Big-$\mathcal{O}$/worst	Fast?
Bubble sort	$\Theta(n^2)$	$\mathcal{O}(n^2)$	No
Selection sort	$\Theta(n^2)$	$\mathcal{O}(n^2)$	No
Insertion sort	$\Theta(n^2)$	$\mathcal{O}(n^2)$	No
Quick sort	$\Theta(n\log(n))$	$\mathcal{O}(n^2)$	Yes
Bucket sort	$\Theta(n)$	$\mathcal{O}(n^2)$	Yes
Heap sort	$\Theta(n\log(n))$	$\mathcal{O}(n\log(n))$	Yes
Merge sort	$\Theta(n\log(n))$	$\mathcal{O}(n\log(n))$	Yes
Radix sort[a]	$\Theta(n)$	$\mathcal{O}(n)$	Yes

Sorting algorithms time complexities to memorize.[b]

[a]Actually Radix is $\Theta(n)$ and $\mathcal{O}(n)$ where k is dependent on the number of bits required to store each key, but the assessment will not make this distinction.

[b]The assessments ask about both worst and average time complexities. They also mistakenly use Big-$\mathcal{O}$ notation for both average and worst time complexities. However, for this list they only differ for *Quicksort* and *Bucket*.

[1]You don't need to know these for the C949 assessment! But any computer science major should be familiar with them.

[2]**Clarification of notation:** The following statements all mean the same thing:
"$f(x)$ is $\mathcal{O}(g(x))$" OR
"$f(x)$ is of $\mathcal{O}(g(x))$" OR
"$f(x) = \mathcal{O}(g(x))$" OR
"$f(x) \in \mathcal{O}(g(x))$"
$\mathcal{O}(g(x))$ is a collection of functions (i.e. a set) so we should write "$f(x) \in \mathcal{O}(g(x))$", but "$f(x) = \mathcal{O}(g(x))$" is commonly used and what you'll see on the assessment. This can be confusing since $\mathcal{O}(n) = \mathcal{O}(n^2)$ but $\mathcal{O}(n^2) \neq \mathcal{O}(n)$!! See this interesting thread on StackExchange Big O Notation "is element of" or "is equal". Note the Wiki cites Donald Knuth.

Sort function	Characteristic
Bubble sort	values are swapped
Bucket sort	values are distributed into sets or "buckets"
Quicksort	look for keywords "pivot"
Merge sort	the list is continually "split"
Radix sort	sorting by least/most significant digits

Sorting algorithms characteristics
to identify from code

Big-$\mathcal{O}$ problems

On the assessment, you will need to determine the Big-$\mathcal{O}$ time-complexity from psuedocode.[a] Answers seem to be limited to $\mathcal{O}(1)$, $\mathcal{O}(n)$, $\mathcal{O}(n^2$, and $\mathcal{O}(n^3)$. A nutshell summary for the assessment:

- If the number of steps stays the same no matter how large, then it's constant time: $\mathcal{O}(1)$
- If you go through a n long list (linearly; say a loop), then you'll take at most n steps: $\mathcal{O}(n)$.
- A loop in a loop will be $\mathcal{O}(n^2)$. A loop in a loop in a loop will be $\mathcal{O}(n^3)$. Often you simply need to count the number of `for` and `while` loops.

[a] Pseudocode on the assessment won't be written as nicely here. Furthermore, it may given in the form of Java, C, or Python.

1 Using the provided pseudo-code, find the worst case performance in Big-$\mathcal{O}$ notation.

Algorithm 1 Some messy pseudo-code

```
procedure SomeProcedure
   for i=1 and 1<=n do
      j=1
      while j<n do
         j=j+2
```

A. $\mathcal{O}(\log n)$ B. $\mathcal{O}(n \log n)$ C. $\mathcal{O}(n^2)$ D. $\mathcal{O}(n)$

2 Using the provided pseudo-code, find the worst case performance in Big-$\mathcal{O}$ notation.

Algorithm 2 Some more messy pseudocode

```
procedure PrintHello
   n=100,000,000
   for i=o; i<n do
      Output "Hello"
      i=i+1
```

A. $\mathcal{O}(100{,}000{,}000)$ B. $\mathcal{O}(n \log n)$ C. $\mathcal{O}(n^2)$ D. $\mathcal{O}(1)$

3 Using the provided pseudo-code, find the worst case performance in Big-$\mathcal{O}$ notation.

Algorithm 3 Some more messy pseudocode

```
procedure SomeProcedure2
    j=0
    for i=o; i<n; i++ do
        j=i+j
```

A. $\mathcal{O}(\log n)$ B. $\mathcal{O}(n \log n)$ C. $\mathcal{O}(n^2)$ D. $\mathcal{O}(n)$

4 Using the provided pseudo-code, find the worst case performance in Big-$\mathcal{O}$ notation. Assume we know that **someMethod(n)** is $\mathcal{O}(\log n)$.

Algorithm 4 Some pseudocode with a method in it

```
procedure SomeProcedure3
    j=0
    for i=o; i<n; i++ do
        j=someMethod(n)
```

A. $\mathcal{O}(\log n)$ B. $\mathcal{O}(n \log n)$ C. $\mathcal{O}(n^2)$ D. $\mathcal{O}(n)$

5 Using the provided pseudo-code, find the worst case performance in Big-$\mathcal{O}$ notation.

Algorithm 5 Some more messy pseudocode

```
procedure SomeProcedure4
    while n> 1 do
        n=n/2
```

A. $\mathcal{O}(\log n)$ B. $\mathcal{O}(n \log n)$ C. $\mathcal{O}(n^2)$ D. $\mathcal{O}(n)$

Key

C, D, D, B, A