Asymptotic Notation

Definition Big O: f(x) is O(g(x)) if

There are positive values c,k such that $0 \leq f(x) \leq cg(x)$ for all x>k.

Advice: Pick values of c and k that are large.

Problem 1) Show that $\sqrt{\boldsymbol{x}}$ is $\boldsymbol{O}(\boldsymbol{x})$.

f(x) is $\sqrt{\boldsymbol{x}}$ g(x) is x

We need to find c and k such that $0 \leq \sqrt{\boldsymbol{x}} \leq cx$ for all x>k.

For big O we pick values that are large.

Let c=2 and k=5

$0 \leq \sqrt{\boldsymbol{x}} \leq 2x$ for all x>5.

We note that

$\sqrt{\boldsymbol{x}}$ is positive for any value greater than 0

2x is positive for any value greater than 0.

$\sqrt{\boldsymbol{x}} \leq 2x$

Square both sides (safe because both sides are positive), then divide by x (positive).

$x \leq (2x)^2$
$x \leq 4x^2$
$1 \leq 4x$
$x > 1/4$

We know that x>5 and 5>1/4, so we can conclude that x>1/4 is true.

Problem 2) Show that 2^x is $O(3^x)$.

f(x) is $\mathbf{2^x}$ g(x) is $\mathbf{3^x}$

We need to find c and k such that $0 \leq \mathbf{2^x} \leq c\mathbf{3^x}$ for all x>k.

For big O we pick values that are large.

Let c=3 and k=5

$0 \leq \mathbf{2^x} \leq 3 \times \mathbf{3^x}$ for all x>5.

We note that

$\mathbf{2^x}$ is positive for any value greater than 0

$\mathbf{3^x}$ is positive for any value greater than 0.

$\mathbf{2^x} \leq 3 \times \mathbf{3^x}$

Take a log of both sides and manipulate the inequality

$\mathbf{2^x} \leq \mathbf{3^{x+1}}$

$\boldsymbol{log}(\mathbf{2^x}) \leq log(\mathbf{3^{x+1}})$

$\boldsymbol{xlog}(\mathbf{2}) \leq (x+1)log(\mathbf{3})$

$\boldsymbol{xlog}(\mathbf{2}) - xlog(3) \leq log(\mathbf{3})$

$\boldsymbol{xlog}(\mathbf{2/3}) \leq log(\mathbf{3})$

$\boldsymbol{x} > \frac{log(\mathbf{3})}{log(2/3)} \approx -2.71$

(The inequality flips because log of 2/3 is negative.)

We know that x>5 and 5>-2.71, so we can conclude that x>$\frac{log(\mathbf{3})}{log(2/3)}$ is true.

Alternate: Sometimes a different choice of c will make the argument easier.

Let c=1 and k=5

$0 \leq \mathbf{2^x} \leq \mathbf{3^x}$ for all x>5.

We note that

$\mathbf{2^x}$ is positive for any value greater than 0

$\mathbf{3^x}$ is positive for any value greater than 0.

$\mathbf{2^x} \leq \mathbf{3^x}$

Take a log of both sides and manipulate the inequality

$\boldsymbol{log}(\mathbf{2^x}) \leq log(\mathbf{3^x})$

$\boldsymbol{xlog}(\mathbf{2}) \leq xlog(\mathbf{3})$

$\boldsymbol{log}(\mathbf{2}) \leq log(\mathbf{3})$

$0.301 \leq 0.477$

Is true.

Definition Big Omega: f(x) is Ω(g(x)) if

There are positive values c,k such that $0 \leq cg(x) < f(x)$ for all x>k.

Advice: Pick values of c that are close to zero and k that are large.

Problem 3) Show that $\boldsymbol{x^4}$ is $\boldsymbol{\Omega(x)}$.

f(x) is $\boldsymbol{x}$**^4** g(x) is x

We need to find c and k such that $0 \leq cx < \boldsymbol{x}$**^4** for all x>k.

For big Ω we pick .

Let c=1/2 and k=10

$0 \leq \frac{1}{2}\boldsymbol{x} \leq x$^4 for all x>10.

We note that

$\frac{1}{2}\boldsymbol{x}$ is positive for any value greater than 0

x^4 is positive for any value greater than 0.

$\frac{\mathbf{1}}{\mathbf{2}}\boldsymbol{x} \leq x^4$

divide by x (positive).

$\frac{\mathbf{1}}{\mathbf{2}} \leq x^3$

We know that x>10 so $x^3 > 10^3$, and since 1000 >$\frac{1}{2}$ we can conclude that $\frac{1}{2} \leq x^3$ is true.

Definition Big Theta: f(x) is Θ(g(x)) if

There are positive values c1, c2, k such that $0 \leq c_1 g(x) \leq f(x) \leq c_2 g(x)$ for all x>k.
Advice: Values c1 and c2 are usually different so do the big O and big Omega proofs separately.

Problem 4) Show that $3x^2 + 10x + 5$ is $O(x^2)$.

Do BigO first:
Goal: $3x^2 + 10x + 5 \leq cx^2$ for all x>k

We will show three things

$3x^2 \leq 3x^2$
$10x \leq 10x^2$
$5 \leq 5x^2$

And then add to get

$\mathbf{3x^2 + 10x + 5 \leq 18x^2}$

Let c=18 and k=5

$3x^2 \leq 3x^2$ for x>5

Divide both sides by x^2 (positive)
$3 \leq 3$ for all x>5.
Is true.

$10x \leq 10x^2$ for x>5

Divide both sides by $10x$ (positive)
$1 \leq x$ for all x>5.
Is true since x>5 and 5>1.

$5 \leq 5x^2$ for x>5

Divide both sides by $10x$(positive)
$1 \leq x$^2 for all x>5.
Is true since x>5 and $x^2 > 25$ and 25>1.

We have shown Big O

Do Big Theta second:
Goal: $cx^2 \leq 3x^2 + 10x + 5$ for all x>k

We will show two things

$2x^2 \leq 3x^2$

$0 \leq 10x + 5$

And then add to get

$2x^2 \leq \mathbf{3x^2 + 10x + 5}$

Let c=2 and k=5

$2x^2 \leq 3x^2$ for x>5

Divide both sides by x^2 (positive)

$2 \leq 3$ for all x>5.

Is true.

$0 \leq 10x + 5$ for x>5

Divide 10 (positive) and subtract

$\mathbf{0 \leq x + 1/2}$

$\frac{-1}{2} \leq x$ for all x>5.

Is true since x>5 and 5>$\frac{-1}{2}$.

We have shown Big Theta

We can now conclude that it is also Big Omega.

Question 5) For the pair of functions $f(x) = x^2$ and $g(x) = 10xlogx$ determine

1) Which function is smaller when x is positive and close to zero.
2) Which function is smaller when x is positive and very large.
3) Find any cross over points for the two functions.

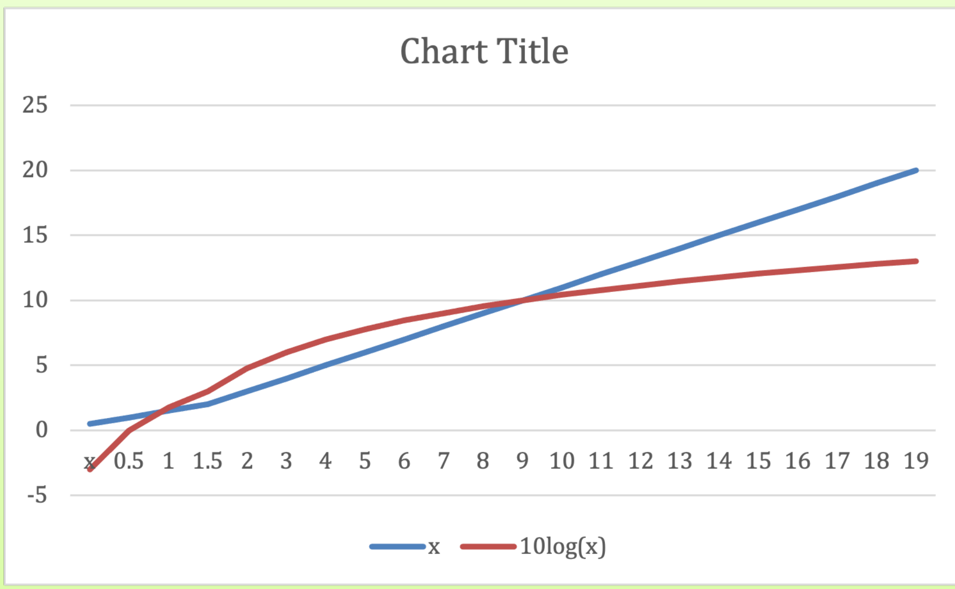
1) Log(x) is negative when x is between 0 and 1. F(x) is always positive, so for small values of x, f(x) is larger than g(x)
2) For very large values of x f(x) larger than g(x)
3) We solve f(x) = g(x) to find any cross over points.

   $$x^2 = 10xlogx$$

   Divide by x (positive)

   $$x = 10logx$$

Since this is non-linear, we will graph to find the cross over points. One is at about 0.7 and the other is at 10.



Question 6) Consider the following algorithm to compute the sum of a list of n digits. Consider two cases: 1) Addition is a constant time operation. 2) Addition is linear in the size of the operands.

```
sum=0
for digit in list_1:
  sum += digit
```

1) Addition is constant time

```
sum=0                          O(1)
for digit in list_1:           O(n) times
  sum += digit                 O(1)
```

The time is O(1) repeated O(n) times is O(n).

2) Addition is linear time

Every time we add into the sum, we may increase the size of the sum by 1. So the size of sum is O(n)

```
sum=0                          O(1)
for digit in list_1:           O(n) times
  sum += digit                 O(n)
```

The time is O(n) repeated O(n) times is O(n^2).

(Note: we can get a tighter upper bound by considering the growth in sum more carefully to get $\boldsymbol{O(nlog(n))}$).

Question 7) Consider the following recursive algorithm to sort a list of numbers of size n. We assume that all the values are limited so comparisons take constant time.

```
def merge (list1, list2):
  merged = []
  while len(list1)!=0 and len(list2)!=0:
       if first = list1[0] < list2[0]:
            merged.append(list1[0])
            list1.remove(0)
       else
            merged.append(list1[0]
            list2.remove(0)
  if len(list0) != 0:
       merged.add(list0)
  else
       merged.add(list1)
  return merged.

def sort (a_list):
  if len(a_list) == 1:
       return a_list
  else:
       mid = len(a_list)/2
       left_sort = sort(a_list[:mid])
       right_sort = sort(a_list[mid:])
       return merge(left_sort, right_sort)
```

We assume that list1 is size O(q) and list2 is size O(p)

```
def merge (list1, list2):
  merged = []                                  O(1)
  while len(list1)!=0 and len(list2)!=0:       O(p+q)
       if first = list1[0] < list2[0]:         O(1)
            merged.append(list1[0])            O(1)
            list1.remove(0)                    O(1)
       else
            merged.append(list1[0]             O(1)
            list2.remove(0)                    O(1)
  if len(list0) != 0:                          O(1)
       merged.add(list0)                       O(q)
  else
       merged.add(list1)                       O(p)
  return merged
```

The time in each loop is constant so the total time is O(p+q) +O(p) + O(q), which is just O(p+q).

```
def sort (a_list):
  if len(a_list) == 1:                         O(1)
       return a_list                           O(1)
  else:
       mid = len(a_list)/2                     O(1)
       left_sort = sort(a_list[:mid])          Recursive n/2
       right_sort = sort(a_list[mid:])         Recursive n/2
       return merge(left_sort, right_sort)O(n/2+n/2)
```

We have a tree of recursive calls and the associated merge work. The work on each level adds to $O(n)$. The number of levels is $O(log_2(n))$, so total is $O(nlog_2(n))$.