Big O Notation

<u>ArrayList in Java</u>

*The size, isEmpty, get, set, iterator, and listIterator operations run in **constant time**.*

*The add operation runs in amortized constant time, that is, adding n elements requires **O(n)** time. All of the other operations run in linear time (roughly speaking).*

- What do they mean by by saying addition is O(N)?

Suppose we consider the following functions that define the timings for different programs:

$t1(n) = 5 + n,$

$t2(n) = 50 + n,$

$t3(n) = 500 +100 n,$

$t4(n) = 5000 +1000 n.$

The impact of the constants is significant for small values of n – say n < 1000000.

But for very large values of n they have little impact.

All of these functions converge, or meet, as n approaches infinity.

- Therefore, all of these functions form a set that are asymptotically dominated by

  $t(n) = n$.

- Form a family of functions of $O(N)$

Similarly we can argue that:

$t(n) = 50N^2$

$t(n) = 10000 + 10N^2$

$t(n) = 1000 + 500N^2$

$t(n) = 100 + 50000N^2$

All belong to the family of functions of $O(N^2)$

Formally

$$O(t(n)) \; is \; O(f(n))$$

$$\text{if } t(n) \leq c * f(n) \text{ for } c > 0 \text{ and } n > n_0$$

Prove t(n) is $O(n^2)$ where

$$t(n) = n^2 + 2 * n + 1$$

Proof:

Must show that $t(n) \le c * n^2$

for constant c and n >= n_0

$$n^2 + 2 * n + 1 \leq n^2 + 2 * n^2 + n^2$$

$$n^2 + 2 * n + 1 \leq 4 * n^2$$

$$c = 4 \text{ and } \mathrm{n}_0 = 1$$

- An easier way to do this is use limits.
- We can prove that f(n) is O(g(n)) by showing that:

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} < \infty$$

- Note

$$\lim_{n\to\infty}\frac{a}{n} = 0, \; a \text{ constant}$$

- To prove that f(n) = 3N is O(N) we simply calculate the limit as N goes to infinity.

$$\lim_{n\to\infty}\frac{f(n)}{g(n)} = \lim_{n\to\infty}\frac{3n}{n} = 3 < \infty$$

- To prove that f(n) = n^2 is not O(n) we show that the limit is not finite.

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} = \lim_{n \to \infty} \frac{n^2}{n}$$

$$= \lim_{n \to \infty} n = \infty$$

Exercise Q1a

$$\lim_{n\to\infty}\frac{f(n)}{g(n)}=\lim_{n\to\infty}\frac{70+50n}{n}$$

$$=\lim_{n\to\infty}\frac{70}{n}+\lim_{n\to\infty}\frac{50n}{n}$$

$$=0+50=50<\infty$$

Summation

$O(1)+O(1)+..+O(1) = k * O(1) = O(1),$

$O(n) + O(n)+..+O(n) = k * O(n) = O(n)$

$O(n) + O(m) = max(O(n), O(m))$

e.g. $O(n^3) + O(n^5) = O(n^5)$

Laws of Big O

Product

$$O(n) * O(n) = O(n^2)$$

$$n * O(n) = O(n^2)$$

$$O(n) * O(m) = O(n * m)$$

$$O(k * f(n)) = k * O(f(n)) = O(f(n))$$

$$O(n^a) * O(n^b) = O(n^{a+b})$$

- The *big-O* sets of order functions form a chain of sub-sets as follows:

$$O(1) << O(\log_2 n) << O(n) << O(n * \log_2 n)$$

$$<< O(n^2) << O(n^k, k > 2) << O(a^n) << O(n!)$$

```
int c = 1; n = 1000;
while(n > c) c=2*c;
//c = 1024
```

$\text{Log}_2 1024 = 10$

$$c = \lceil \log n \rceil$$

```
static long power1(int a, int b){
 long z = 1; int k = 0;
 while(k < b){
      z = z * a;
      k = k + 1;
 }
 return z;
}
```

$t(n) = c1 + c2*b$

```
static long power2(int a, int b){
   int c = 1; int s = b; long z = 1;
  while(b >= c) c=2*c;
  while(c != 1){
  c = c/2; z = z * z;
  if(s >= c){
       s = s - c; z = z * a;
  }
   }
   return z;
 }
```

t(n) = c1+c2*logb + c3*logc

Show that the cost function t(n) for the code fragment

```
int f[][] = new int[n][n];
for(inti = 0; i < n; i++)
    for(int j = 0; j < n; j++)
        f[i][j] = i*j;
```

is

$$t(n) = k * \sum_{i=0}^{n-1} n$$

Let k = cost of all assignments

t(n) = k*n+k*n+...+k*n

= k*(n+n+...+n)

=

$$k * \sum_{i=0}^{n-1} n$$

```
int i = 0;
while(i < n){
    int j = i;
    while(j < n){
     p(j) //where p(j) does not involve a loop
     j++;
    }
    i++;
}
```

Show that $$t(n) = k * \sum_{i=0}^{n-1} (n - i) = k * \frac{n * (n+1)}{2}$$

Let cost of all assignments and p(j) be k

t(n) = k*n + k*(n-1)+k*(n-2)+...+k*2+k*1

= k*[n + (n-1) + (n-2) + ... + 2 + 1]

=

$$k * \sum_{i=0}^{n-1} (n-i)$$

$$= k * \frac{n*(n+1)}{2}$$

$$\sum_{i=0}^{n-1}(n-i) = \sum_{i=0}^{n-1} n - \sum_{i=0}^{n-1} i$$

$$= n^2 - \frac{(n-1)*n}{2} = \frac{2n^2 - n^2 + n}{2} = \frac{n^2 + n}{2}$$

Show that the cost of code fragment:

```
int sum = 0;
for(int j = 1; j <= n; j = j * 2)
   for(int i = 0; i < n; i++)
      sum = sum + 1;
```

is

$$t(n) = k * \sum_{j=1}^{\lceil \log_2 n \rceil} n$$

Outer loop executes $\lceil \log_2 n \rceil$ times

t(n) = k*n + k*n +... + k*n

= k*(n + n + ... + n)

= $$k * \sum_{j=1}^{\lceil \log_2 n \rceil} n$$

= $$k * n * \lceil \log_2 n \rceil$$

big-Oh	t(n)	Name
$O(1)$	$t(n) = k$, a constant	constant time
$O(\log_2 n)$	$t(n) = a + b * \log_2 n$	logarithmic time
$O(n)$	$t(n) = a + b * n$	linear time
$O(n * \log_2 n)$	$t(n) = a + b * n * \log_2 n$	n log n
$O(n^2)$	$t(n) = a + b * n^2$	quadratic time
$O(n^3)$	$t(n) = a + b * n^3$	cubic time
$O(2^n)$	$t(n) = a + b * 2^n$	exponential time
$O(n!)$	$t(n) = a + b * n!$	factorial time

$$
\begin{aligned}
O(t(n)) &= O(a + b*n*\log_2 n) \\
&= O(a) + O(b*n*\log_2 n) \\
&= a*O(1) + b*O(n*\log_2 n) \\
&= O(1) + O(n*\log_2 n) \\
&= O(n*\log_2 n)
\end{aligned}
$$

```
public static void main(String args[]){
  int f[][] = new int[N][N];
  int g[][] = new int[N][N];
  int add[][] = new int[N][N];
```

$O(a)$

```
// init both matrices
for(int i = 0; i < f.length;i++){
  for(int j = 0; j < f[0].length; j
++){
      f[i][j] = (int)
(Math.random()*10);
      g[i][j] = (int)
(Math.random()*10);
      }
  }
```

*O(f.length) **

*O(b*f[0].length)*

<pre>// add corresponding elements on a row by row basis
for(int i = 0; i < f.length; i++){
    for(int j = 0;j < f[0].length; j++){
        add[i][j] = f[i][j] + g[i][j];
    }
}
}</pre>
	$O(f.length) * O(c*f[0].length)$

$$O(t(n)) = O(a) + O(f.length) * O(b * f[0].length)$$
$$+ O(f.length) * O(c * f[0].length)$$

$$= O(a) + O(n) * O(b * n) + O(n) * O(c * n),$$
$$\text{where } n = f.length = f[0].length$$

$$= O(1) + O(n^2) + O(n^2)$$
$$= O(n^2)$$

<table>
<tr><th><pre>static void doublingUpTo(int n){
  int p = 0; int c = 1; int s = n;</pre></th><th>$O(1)$</th></tr>
<tr><td><pre>while(n >= c) c = c * 2;</pre></td><td>$O(\log_2 n)$</td></tr>
<tr><td><pre> while(c != 1){
   c = c / 2; p = p * 2;
   if(s >= c){
      p = p + 1; s = s - c;
   }
 }
 // p == n
}</pre></td><td>$O(a+b*\log_2 n)$</td></tr>
</table>

$$
\begin{aligned}
O(d(n)) &= O(1) + O(\log_2 n) + O(a + b * \log_2 n) \\
&= O(1) + O(\log_2 n) + O(a) + O(b * \log_2 n) \\
&= O(1) + O(\log_2 n) + O(\log_2 n) \\
&= O(\log_2 n)
\end{aligned}
$$

- notation provides a way to classify algorithms that can be used for comparison purposes.

- A program that has *O(1)* will perform better than a program of *O(n)*, in general.

- It does not provide comparison for algorithms in the same class.

- We can also extend this idea to analyzing *best case, average case* and *worse case* scenarios for given algorithm.

LinearSearch

```
static boolean search(int f[], int x){
    boolean found = false;
    int j = 0;
    while(j < f.length && ! found){
        if(f[j] == x) found = true;
        else j++;
    }
    return found;
}
```

- **Worst case**: x not present O(N)
- **Best case**: x is first element O(1)
- **Average case**: Suppose probability of searching for any element is the same. Then on average the cost of searching is:

$$\left(\sum_{i=0}^{n} i\right) / n = \frac{n*(n+1)}{2*n} = \frac{n+1}{2} = O(n)$$

Exercise Q1a

$$\lim_{n\to\infty}\frac{f(n)}{g(n)}=\lim_{n\to\infty}\frac{70+50n}{n}$$

$$=\lim_{n\to\infty}\frac{70}{n}+\lim_{n\to\infty}\frac{50n}{n}$$

$$=0+50=50<\infty$$

Exercise Q1b

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{20 + 45 * n + 5 * n^2}{n^2}$$

$$= \lim_{n\to\infty} (\frac{20}{n^2} + \frac{45 * n}{n^2} + \frac{5 * n^2}{n^2})$$

$$= 0 + 0 + 5) = 5 < \infty$$

Question

Prove $O(10+20*N+3*N^2) = O(N^2)$

$$
\begin{aligned}
& O(10+20*N+3*N^2) \\
&= O(10)+O(20*N)+O(3*N^2) \\
&= O(1)+O(N)+O(N^2) \\
&= O(N^2)
\end{aligned}
$$