Analysis of Algorithms

It is convenient to have a measure of the amount of work involved in a computing process, even though it be a very crude one ... We might, for instance, count the number of additions, subtractions, multiplications, divisions, recordings of numbers, ...

Sum of first N natural numbers

```
static long sumN(long n){
    long s = n*(n+1)/2;
    return s;
}
```

```
static long sumN1(long n){
    long s = 0;
    for(int j=0; j < n; j++) s=s+(j+1);
    return s;
}
```

Rules of Indices

$$a^m * a^n = a^{m+n}$$

$$a^m \div a^n = a^{m-n}$$

$$a^0 = 1$$

$$a^{\frac{1}{n}} = \sqrt[n]{a}$$

$$(a^n)^m = a^{n*m}, \qquad \frac{1}{a^n} = a^{-n}$$

Def. Log

$$\log_b x = y \equiv b^y = x$$

$$log_{10}100 = 2 \equiv 10^2 = 100$$

$$log_{10}1000 = 3 \equiv 10^3 = 1000$$

$$log_{10}1 = 0 \equiv 10^0 = 1$$

$$log_{10}165 = 2.2175 \equiv 10^{2.2175} = 165$$

$$\log(x * y) = \log(x) + \log(y)$$

$$\log(x \div y) = \log(x) - \log(y)$$

$$\log(x^n) = n * \log(x)$$

Evaluate each of the following:

$$log_2 16, \qquad log_2 1024, \qquad log_2 \frac{1}{2}$$

Solution

$$log_2 16 = log_2 2^4 = 4 * log_2(2) = 4 * 1 = 4$$

$$log_2 1024 = log_2 2^{10} = 10 * log_2(2) = 10 * 1 = 10$$

$$log_2 \frac{1}{2} = log_2 2^{-1} = -1 * log_2(2) = -1 * 1 = -1$$

Show that $\log(n!) = \log(1) + \log(2) + .. + \log(n)$

Solution

$$\log(n!) = \log(n * (n-1) * .. * 2 * 1) = \log(1) + \log(2) + .. + \log(n)$$

Changing Base

$$log_b x = \frac{log_c x}{log_c b}$$

$$log_8 128 = \frac{log_2 128}{log_2 8} = \frac{log_2 2^7}{log_2 2^3} = \frac{7}{3}$$

Show log360 = 3log2+2log3+log5

$$\log(360)$$

$$= \log(5 * 8 * 9)$$

$$= \log(5) + \log(8) + \log(9)$$

$$= \log(5) + \log(2^3) + \log(3^2)$$

$$= \log(5) + 3\log(2) + 2\log(3)$$

- Sequence
  - Set of terms that are related in some way

    2, 4, 6, 8, 10,..

    1, 4, 9, 16, 25, ..
- Series
  - Sum of a sequence of terms
  - 1 + 3 + 5 + 7 + 9 + ..

Arithmetic Progression

$$T(n) = a + (n - 1) * d$$

$$S(n) = n/2*\{2 * a + (n - 1) * d\}$$

Examples

1, 4, 7, 10,

$$T(n) = 1 +(n - 1) * 3 = 1 + 3*n - 3 = 3*n - 2$$

$$T(20) = 3 * 20 - 2 = 58$$

$$S = 2 + 5 + 8 + 11 + ..$$

$$S(n) = n/2 *\{2*2 + (n - 1) * 3\}$$
$$= n/2 *(4 + 3*n - 3)$$
$$= n*(3*n + 1)/2$$

$$S(4) = 4*(3*4 + 1)/2$$
$$= 4*(13)/2$$
$$= 26$$

$$\sum_{i=1}^{n} term$$

Read as: *the sum from i equals 1 to n of term*.

$$\sum_{i=1}^{10} i = 1 + 2 + 3 + .. + 9 + 10 = 55$$

$$\sum_{i=1}^{n} 1 = 1 + 1 + .. + 1 = n * 1 = n$$

$$\sum_{i=1}^{n} i = 1 + 2 + .. + (n-1) + n = n * (n+1)/2$$

$$\sum_{i=1}^{n} k * i = k * \sum_{i=1}^{n} i = k * n * (n+1)/2$$

$$\sum_{i=1}^{n} i^2 = 1^2 + 2^2 + .. + n^2 = n * (n+1) * (2 * n + 1)/6$$

$$\sum_{i=1}^{n}(n-i) = n*(n+1)/2$$

Solution

$$\sum_{i=0}^{n}(n-i) = n+(n-1)+..+2+1+0 = n*(n+1)/2$$

Show that $\sum_{k=1}^{n}(2 * k - 1) = n^2$

$$\sum_{k=1}^{n}(2 * k - 1)$$

$$= 2 * \sum_{k=1}^{n} k - \sum_{k=1}^{n} 1$$

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

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

- Logs are relevant in computing because they are used in the analysis of algorithms particularly in relation to what are termed divide and conquer algorithms, e.g. binary searching.
- The log function grows very slowly and, hence, log scales are often used to compress large scale scientific data.

Richter Scale

- Quake of 7.0 is 1000 times greater than one of 4.0

$$logQ1 = 7 \equiv Q1 = 10^7$$

$$logQ2 = 4 \equiv Q2 = 10^4$$

$$Q1 = 10^7 = 10^3 * 10^4 = 1000 * Q2$$

Compare Linear Sequence with Log Sequence

Linear sequence:

10,20, 30, 40, 50, 60, 70, 80, 90, 100, 110, 120, 130, 140, 150, 160, 170, 180, 190, 200

Common Logs:

1.000, 1.301, 1.477, 1.602, 1.699, 1.778, 1.845, 1.903, 1.954, 2.000, 2.041, 2.079, 2.114, 2.146, 2.176, 2.204, 2.230, 2.255, 2.279, 2.301

Compare Linear Sequence with Log_2

Linear sequence:

10,20, 30, 40, 50, 60, 70, 80, 90, 100, 110, 120, 130, 140, 150, 160, 170, 180, 190, 200

Logs base 2:

3.322, 4.322, 4.907, 5.322, 5.644, 5.907, 6.129, 6.322, 6.492, 6.644, 6.781, 6.907, 7.022, 7.129, 7.229, 7.322, 7.409, 7.492, 7.570, 7.644

Measuring Time

- 1sec = 10^3ms (miliseconds)
- 1sec = 10^6 μs (microseconds)
- 1sec = 10^9ns (nanoseconds)

Calculating Running Times on HAL

Statement	Unit cost (ns)
`-, *, /, %, <, >, ==, >=, <=, !=, =`	*10ns*
Function invocation	*50ns*
Argument passing	*10ns* per argument
`return`	*50ns*
`if(b) s1; else s2`	the cost of `b` plus the max cost of `s1, s2`
`for, while loops`	*totalCost = cost of initialization of variables* + *(n+1) * cost of evaluating guard on loop* + *n * cost of executing loop body,* *where n equals the number of iterations of the loop.*
`new`	*100ns*
Calculating array indices	*50ns*

cost of executing:

- +, -, *, /, %, <, >, ==, >=, <=, !=, =,

be *10ns*

Running Times on HAL

Code	Unit cost (ns)	Total cost (ns)
x = x + 1	10,10	20
x == 2 || x == 7	10, 10, 10	30

x = x + y;
y = x - y;
y = x - y;
	10, 10
	10, 10
	10, 10
		60

```
static long sumN(long n){    50 + 10
    long s = n*(n+1)/2;      10 + 10 + 10 + 10
    return s;                50
}
```

The time taken to execute an if statement of the form if(b) s1; else s2; is the cost of b plus the max cost of s1, s2. For example, the cost of executing

```
if(x > 0) x = 1; else x=x*x;
```

is *30ns* because the boolean expression costs *10ns* and the max cost of *(10ns,20ns)* is *20ns*.

totalCost = cost of initialization of variables

+

*(n+1) * cost of evaluating guard on loop*

+

*n * cost of executing loop body,*

where n equals the number of iterations of the loop.

```
long s = 0;
for(int j = 0; j < 1000; j = j + 1)
        s = s + (j + 1);
```

$$totalCost = (10 + 10) + 1001 * 10 + 1000*(30+20)$$

$$= 20 + 10010 + 50000$$

$$= 60030ns$$

```
static long sumN1(long n){
    long s = 0;
    for(int j=0; j < n; j++) s=s+(j+1);
    return s;
}
```

totalCost =

(50 + 10) - invocation, parameter

+ (10+10) - initialization of variables

*+ (n+1)*10 – evaluation of loop guard*

*+ n *50 - loop body*

+ 50 - return value

totalCost =

$$60 + 20 + (n + 1) * 10 + n * 50 + 50$$

$$= (140 + 60 * n) \text{ ns}$$

N = 1000000

*Cost = 140 + 60*1000000 ns*

*= 6*10000microseconds*

= 60 milliseconds

= 0.06seconds

Nested loops

```
int f[][] = new int[n][n];
int a = 0;
while(a < n){
    int b = 0;
    while(b < n){
        f[a][b] = 1;
        b = b + 1;
    }
    a = a + 1;
}
```

	Cost
int f[][] = new int[n][n];	*100*
int a = 0;	*10*
while(a < n){	*(n+1)*10*
int b = 0;	
while(b < n){	*n*K*
f[a][b] = 1;	
b = b + 1;	
}	
a = a + 1;	
}	

$$T(n) = 100+10+(n+1)*10+n*K$$
$$= 120 + 10*n + n*K$$

$$K(n) = 10+(n+1)*10 + n*(50+10+20) + n*10$$
$$= 20 + 100*n$$

$$T(n) = 120 + 10*n + n*(20 + 100*n)$$
$$= 120 + 30*n + 100*n^2$$

For n = 10000

$$T(10000) = 120 + 30*10000 + 100*10000*10000$$

$$= 10 \text{ seconds}$$

Benchmarking

```
long start = System.nanoTime();

 //function to test

 long end = System.nanoTime();
 long k = end-start;
 System.out.println("Time in nanseconds:
                                        "+k);
```

Find sum of the first n terms of the series:

1 + 3 + 6 + 10 + 15 + ..

1 + (1+2) + (1+2+3) + (1+2+3+4) + ..

```
static long sumS1(int n){
    long k = 0;
    for(int j = 0; j < n; j++)
        k = k + sumN(j+1);
    return k;
}
```

T(n) = 140 + 200*n, sumN(n) = 150ns

```
static long sumS2(int n){
    long k = 0;
    for(int j = 0; j < n; j++)
        k = k + sumN1(j+1);
    return k;
}
```

$T(n) = 130 + 160*n + 30*n^2$

(sumN1(n) = 130+30*n)

$$\sum_{i=1}^{n}\left(\sum_{j=1}^{i} j\right) = 1 + (1+2) + (1+2+3) + ..$$

By expanding this definition we derive a formula to solve the problem. We know that the sum of *j* from *1* to *i* is *i*(i+1)/2*. That is,

$$\sum_{j=1}^{i} = i * (i+1)/2$$

This means that

$$\sum_{i=1}^{n}\left(\sum_{j=1}^{i} j\right) = \sum_{i=1}^{n} i * \frac{i+1}{2} = \frac{1}{2}\sum_{i=1}^{n}(i^2 + i) = \frac{1}{2}\sum_{i=1}^{n} i^2 + \frac{1}{2}\sum_{i=1}^{n} i \ .$$

This gives

$$\frac{1}{2}\left[\frac{n*(n+1)*(2n+1)}{6}+\frac{n*(n+1)}{2}\right]$$

$$= n*(n+1)*(n+2)/6$$

```
static long sumS3(int n){
    long k = n*(n+1)*(n+2)/6;
    return k;
}
```

For n = 10000, sumS3 is 10000 faster than sumS1 and 10 million times faster than sumS2

Where do logs come in?

```
int k = 1024;
while(k > 0){
    //loop body here
    k = k/2;
 }
```

What is cost of executing loop?

$k = 1024 = 2^{10}$

Each iteration divides k by 2

Therefore, number of iterations = 10.

Observe $\log_2 1024 = 10$

$$\text{Log}_2 1024 = \log_2 2^{10}$$

$$= 10 * \log_2 2 = 10 * 1 = 10$$

- What if k not exact power of two?

- Number of iterations =

$$\lfloor \log_2 k \rfloor$$

- For example,

$$\lfloor \log_2 2000 \rfloor = 10$$

- Binary search is:

$$\lfloor \log_2 N \rfloor, \text{where N} = \text{size of data set}$$

Binary Search

```
static boolean binSearch(int f[],int x){
    int j = 0; int k = f.length;
    while(j + 1 != k){
        int i = (j + k)/2;
        if( x >= f[i]) j = i;
        else  k = i;
    }
    return(f[j]==x);
}
```

$$t(n) = 200 + 20 * log_2(n+1) + 100 * log_2 n$$

Ignoring the constant 200 and the additional +1, we get:

$$t(n) = 120 * log_2 n$$

Supposing n = 1Mb we would get a running time of:

$$t(2^{20}) = 120 * log_2(2^{20}) = 120 * 20 * log_2 2 = 120 * 20 * 1 = 2400ns$$