综上所述，等比数列$\{a_n\}$的求和公式为：

$$S_n = \begin{cases} \frac{a_1 - a_1 q^n}{1-q} & q \neq 1 \\ n a_1 & q = 1 \end{cases}$$

经过推导，可以得到另一个求和公式：当$q \neq 1$时，

$$S_n = \frac{a_1(1-q^n)}{1-q} = \frac{a_1 q^n - a_1}{q-1}$$

first

ratio

当-1<q<1时，等比数列无限项之和

由于当$-1 < q < 1$及n的值不断增加时，q^n的值便会不断减少而且趋于0，因此无限项之和为：

$$S = \lim_{n\to\infty} S_n = \lim_{n\to\infty} \frac{a_1 - a_1 q^n}{1-q} = \frac{a_1}{1-q}$$

Example 5

```
read(n) ; a=0.7 ;
while( n>1) {
    For( j=1; j<n ; j++ ) print("*") ;
    n=a*n ; }
```

- The **first** time the outer loop is called, the "print" is called n times.
- The **2nd** time the outer loop is called, the "print" is called an times.
- The **3rd** time the outer loop is called, the "print" is called $a^2 n$ times…
- The **k' th** time the outer loop is called, the "print" is called $a^k n$ times

- Let t be the number of iterations of the outer loop. Then the total time

$$= n + an + a^2 n + a^3 n + \dots a^t n =$$
$$n(1 + a + a^2 + a^3 + \dots a^t) <$$
$$n(1 + a + a^2 + a^3 + \dots a^t + \dots)$$
$$= n / (1-a) = O(n).$$

- Same analysis holds for any $a<1$

$$\Omega(g(n)) = \{\text{funcs that grow at least as fast as } g(n)\}$$

$$\Omega(g(n)) = \left\{ f(n) : \begin{array}{l} \exists c>0, n_0>0 \\ \forall n \geq n_0 \\ c\, g(n) \leq f(n) \end{array} \right\}$$

Examples 4

TimeComplexity : Outer n. Inner O(n) . Total : O(n²)

```
read(n) ;
for(i=1 ; i < n ; i++)
  for( j=i ; j <n ; j += i )
     print( "*" ) ;
```

- More "sensitive" analysis:
- For $i=1$ we run through $j=1,2,3,4\dots n$, total n times.
- For $i=2$ we run through $j=2,4,6,8,10\dots n$, total $n/2$ times.
- For $i=3$ we run through $j=3,6,9,12\dots n$, total $n/3$ times .
- For $i=4$ we run through $j=4,8,12,16\dots n$, total $n/4$ times.
- For $i=n$ we run through $j=n$, total $n/n=1$ times.
- Summing up: $T(n)=n+n/2+n/3+n/4+\dots n/n =$
  $n(1+1/2+1/3+1/4+\dots 1/n) \approx n \ln n$
  Harmonic Sum

Properties of big-O

- **Claim:** if $T_1(n)=O(g_1(n))$ and $T_2(n)=O(g_2(n))$ then
  $T_1(n)+T_2(n)=O(g_1(n) + g_2(n))$
- **Example**: $T_1(n)=O(n^2)$, $T_2(n)=O(n \log n)$ then
  $T_1(n)+T_2(n)=O(n^2 + n \log n) = O(n^2)$
- **Proof:** We know that there are constants n_1, n_2, c_1, c_2 **s.t.**
  - for every $n>n_1$ $T_1(n) < c_1 g_1(n)$. (definition of big-O)
  - for every $n>n_2$ $T_2(n) < c_2 g_2(n)$. (definition of big-O)
  - Now set $n' = \max\{n_1, n_2\}$, and $c' = c_1 + c_2$, then
    - for every $n>n'$ we have that
    - $T_1(n)+T_2(n) < c_1 g_1(n) + c_2 g_2(n) \leq$
      $c' g_1(n) + c' g_2(n) =$
      $c' (g_1(n) + g_2(n))$

More properties of big-O

- **Claim:** if $T_1(n)=O(g_1(n))$ and $T_2(n)=O(g_2(n))$ then
  $T_1(n)\, T_2(n)=O(g_1(n)\, g_2(n))$
- **Example:** $T_1(n)=O(n^2)$, $T_2(n)=O(n \log n)$ then
  $T_1(n)\, T_2(n)=O(n^3 \log n)$
- Similar properties hold for Θ

The lower bound trick – Second example

We are about to insert n keys $\{k_1, \dots k_n\}$ into an empty AVL tree.
How much time would it take ?

Upper bound: When the $i+1$ key is inserted, the tree contains i keys, so its height is O(log i), and an insert operation takes O(log i) which is also O(log n)

So the overall running time is
O(log 1) + O(log 2) + O(log 3) +O(log 4)+....+O(log n) ≤
O(log n) + O(log n) + O(log n)+O(log n)+...++O(log n) = O(n log n)

This is an upper bound. What is the lower bound ?
- Ω(n) ?
- Ω(n+1) ?
- Ω(2^n) ?
- Ω(n log n)

The lower bound trick – a less trivial example

We **demonstrate** this trick by giving an $\Omega(n \log n)$ bound on the time $T(n)$ required to insert n keys into an (initially empty) balanced search tree.

The i'th insertion takes $K \log(i)$ time (for a constant K, that we ignore). Hence

$$\sum_{i=1}^{n} \log i =$$
$$\log 1 + \log 2 \cdots + \log(\tfrac{n}{2} - 1) + \log(\tfrac{n}{2}) + \log(\tfrac{n}{2} + 1) + \cdots + \log n$$
$$\geq \log(\tfrac{n}{2}) + \log(\tfrac{n}{2} + 1) + \log(\tfrac{n}{2} + 2) + \cdots + \log n$$
$$\geq \log(\tfrac{n}{2}) + \log(\tfrac{n}{2}) + \log(\tfrac{n}{2}) + \cdots + \log(\tfrac{n}{2}) =$$
$$= (\tfrac{n}{2}) \log(\tfrac{n}{2}) = \Omega(n \log n)$$

eliminated small part