Example

```
for (i = 1; i < n; i++)
  for (j = i; j < n; j++)
    x++;
```

Example

```
while (n>1)
 {   for ( int j = 0; j < n; j++)
        x++;
      n /= 2;
 }
```

- n + n/2 + n/4 + … 1
- Geometric progression
- What is the total?

Recursive Examples

```
void doit(int n)
{  if (n<=1) return;
   for (int i=0; i < n; i++)
         x = x + 1;
   doit(n/2);
   doit(n/2);
}
```

Recursive Examples

```
void doit(int n)
{  if (n<=1) return;
   for (int i=0; i < n; i++)
      x = x + 1;
   doit(n/2);
}
```

Recursive Examples

```
void doit(int n)
{  if (n==1) return;
    x++;
    doit(n/2);
}
```

Recursive Examples

```
void doit(int n)
{  if (n<=1) return;
    x++;
   doit(n/2);
   doit(n/2);
}
```

What is the complexity?

```
for (i=0; i < n; i++)
  a[i] = 0;
for (i=0; i < n; i++)
   for (j=0; j < n; j++)
     a[i] += a[j] + i + j;
```

What is the complexity?

```
if (zeroOut)
     for (i=0; i < n; i++)
        a[i] = 0;
else
   for (i=0; i < n; i++)
      for (j=0; j < n; j++)
        a[i] += a[j] + i + j;
```

Other bounds

Name	Expression	Growth Rate	Similar to
Big-Oh	$T(N) = O(F(N))$	Growth of T(n) is $\leq$ growth of F(n)	**Less than or Equal (Upper bound)**
Big Omega	$T(N) = \Omega(F(N))$	Growth of T(n) is $\geq$ growth of F(n)	**Greater than (lower bound)**
Big-Theta	$T(N) = \Theta(F(N))$	Growth of T(n) is = growth of F(n)	**Equal to (tight bound)**
Little-Oh	$T(N) = o(F(N))$	**Growth of T(n) is < growth of F(n)**	**T(n) is negligible compared to F(n)**

Why don't we just use Big Theta? Isn't a tight bound best?

Def : A lower bound of a problem is the least time complexity required for any algorithm which can be used to solve this problem (for any input).

It is difficult to prove that no better algorithm can be found.

When recursive problems are regular in nature, we can use a Formula Approach

- Theorem: Assume $T(n) = a(T(n/b))+O(n^k)$ is the time for the function.
  - If $a > b^k$, the complexity is $O(n^{\log_b a})$.
  - If $a = b^k$, the complexity is $O(n^k \log n)$.
  - If $a < b^k$, the complexity is $O(n^k)$.

  a is number of recursive calls at one level

  b is how size is divided between calls

  k is amount of work as an exponent (# of for loops)

Study the table below which compares various complexities. Note that even for small *n* (1000), time is measured in years for 2^n.
Complexity 2^n is termed intractable.

Log n	*n*	n log n	n^2	n^3	2^n
0	1	0	1	1	2
1	2	2	4	8	4
2	4	8	16	64	16
3	8	24	64	512	256
4	16	64	256	4096	65,536
5	32	160	1024	32,788	4,294,967,296

Determining Complexity from Experimental Evidence

n	T(n)
2	10
4	10
8	10
16	11
32	8

n	T(n)
2	10
4	17
8	32
16	66
32	130