1> N-squared = N^2 N factorial = N! ascending order 升序

2> The Dictionary Principle: Turning an O(N) problem into an O(log N) problem.

3> Expression: ! > && = || > * = / = % > + = - 4> Linear: n Linearithmic:nlgn

5> Union-find:

Algorithm	Initialize	Union	Find	Connected
Quick-find	N	N	1	1
Quick-union	N	lgN+ ~ N+	lgN ~ N	lgN ~ N
Weighted QU	N	lgN+	lgN	lgN
QU + PC	N	1+ ~ N+	1 ~ N	1 ~ N
WQU + PC	N	lg*N+	lg*N	lg*N

(+: includes cost of finding roots)

6> Sort:

Selection sort:

```
int N = a.length;
for (int i = 0; i < N; i++) {
    int min = i;
    for (int j = i+1; j < N; j++)
        if (less(a[j], a[min]))
            min = j;
    swap(a, i, min);
}
```

Insertion sort:

```
int N = a.length;
for (int i = 0; i < N; i++)
    for (int j = i; j > 0; j--)
        if (less(a[j], a[j-1]))
            swap(a, j, j-1);
        else break;
```

Shell sort:

```
int N = a.length;
int h = 1;
while (h < N/3) h = 3*h + 1; // 1, 4, 13, 40, ...
while (h >= 1) { // h-sort the array.
    for (int i = h; i < N; i++) {
    for (int j = i; j >= h && less(a[j], a[j-h]); j -= h)
        swap(a, j, j-h);
    }
    h = h/3;}
```

Merge sort:

```
private static void merge(Comparable[] a,
Comparable[] aux, int lo, int mid, int hi) {
for (int k = lo; k <= hi; k++)
    aux[k] = a[k];
    int i = lo, j = mid+1;
    for (int k = lo; k <= hi; k++) {
        if (i > mid) a[k] = aux[j++];
        else if (j > hi) a[k] = aux[i++];
        else if (less(aux[j], aux[i])) a[k] = aux[j++];
        else a[k] = aux[i++];
    } }
```

```
private static void sort(Comparable[] a, Comparable[] aux, int
lo, int hi) {
    if (hi <= lo) return;
    int mid = lo + (hi - lo) / 2;
    sort(a, aux, lo, mid);
    sort(a, aux, mid+1, hi);
    merge(a, aux, lo, mid, hi); }
```

Quick sort:

```
private static void sort(Comparable[] a, int lo, int
hi) { if (hi <= lo) return; int j = partition(a, lo, hi);
sort(a, lo, j-1); sort(a, j+1, hi); }
```

```
private static int partition(Comparable[] a, int lo, int hi) {
    int i = lo, j = hi+1;
    while (true) {
        while (less(a[++i], a[lo])) if (i == hi) break;
        while (less(a[lo], a[--j])) if (j == lo) break;
        if (i >= j) break;
        swap(a, i, j);
    }
    swap(a, lo, j); return j;}
```

	In-place	Stable	Best	Average	Worst	remarks
Selection	Y		1/2 N^2	1/2 N^2	1/2 N^2	N exchanges
Insertion	Y	Y	N	1/4 N^2	1/2 N^2	Use for small N or partially ordered
Shell	Y		N $\log_3$ N	?	c $N^{3/2}$	Tight code; subquadratic
Merge		Y	1/2 N lg N	N lg N	N lg N	N lg N guarantee; stable
Timsort		Y	N	N lg N	N lg N	Improve mergesort when preexisting order
Quick(3 Way)	Y		N lg N	1.39 N lg N	1/2 N^2	Fastest in practice

7> Use Linked List to implement Bag, Queue and Stack. 8> Stirling's approximation for N! = lg N! = ~NlgN

9> An inversion is a pair of keys that are out of order.

An array is partially sorted if the number of inversions are <= CN.

Number of exchanges equals the number of inversions. Number of compares = exchanges + (N - 1).

```
private static void sort(Comparable[] a, int lo, int hi) { if (hi <= lo) return; int lt = lo, gt = hi; Comparable v = a[lo]; int i = lo;
while (i <= gt) { int cmp = a[i].compareTo(v); if (cmp < 0) swap(a, lt++, i++); else if (cmp > 0) swap(a, i, gt--); else i++; }
sort(a, lo, lt - 1); sort(a, gt + 1, hi); }
```