插入排序

最差时间复杂度

$O(n^2)$

最优时间复杂度

$O(n)$

平均时间复杂度

$O(n^2)$

最差空间复杂度

总共$O(n)$，需要辅助空间$O(1)$

插入排序（英语：Insertion Sort）是一种简单直观的排序算法。它的工作原理是通过构建有序序列，对于未排序数据，在已排序序列中从后向前扫描，找到相应位置并插入。**插入排序**在实现上，通常采用in-place排序（即只需用到O(1)的额外空间的排序），因而在从后向前扫描过程中，需要反复把已排序元素逐步向后挪位，为最新元素提供插入空间。

记载

最早拥有排序概念的机器出现在1901至1904年间由Hollerith发明出使用基数排序法的分类机，此机器系统包括打孔，制表等功能，1908年分类机第一次应用于人口普查，并且在两年内完成了所有的普查数据和归档。 Hollerith在1896年创立的分类机公司的前身，为电脑制表记录公司（CTR）。他在电脑制表记录公司（CTR）曾担任顾问工程师，直到1921年退休，而电脑制表记录公司（CTR）在1924年正式改名为IBM。

算法描述

一般来说，**插入排序**都采用in-place在数组上实现。具体算法描述如下：

1. 从第一个元素开始，该元素可以认为已经被排序
2. 取出下一个元素，在已经排序的元素序列中从后向前扫描
3. 如果该元素（已排序）大于新元素，将该元素移到下一位置
4. 重复步骤3，直到找到已排序的元素小于或者等于新元素的位置
5. 将新元素插入到该位置后
6. 重复步骤2~5

如果比较操作的代价比交换操作大的话，可以采用二分查找法来减少比较操作的数目。该算法可以认为是**插入排序**的一个变种，称为二分查找插入排序。

范例程式码

C语言

```
void insertion_sort(int arr[], int len) {
        int i, j;
        int temp;
        for (i = 1; i < len; i++) {
                temp = arr[i]; //與已排序的數逐一比較，大於temp時，該數向後移
                for (j = i - 1; j >= 0 && arr[j] > temp; j--) //j循环到-1时，由于[[短路求值]](http://zh.wikipedia.org/wiki/短路求值)，不会运算array[-1]
                        arr[j + 1] = arr[j];

                arr[j+1] = temp; //被排序数放到正确的位置
        }
}
```

C++

```
template<typename T> //整數或浮點數皆可使用,若要使用物件(class)時必須設定大於(>)的運算子功能
void insertion_sort(T arr[], int len)
{   int i, j, temp;
        for (i = 1; i < len; i++)
    {  temp = arr[i];
           for (j=i-1; j>=0 && arr[j]>temp; j--)
         arr[j+1] = arr[j];

         arr[j+1] = temp;
        }
}
```

C#

```
public static void InsertSort(double[] data) {
        int i, j;
        var count = data.Length;
        for (i = 1 ; i < count ; i++) {
                var t = data[i];
                for(j = i - 1; j >= 0 && data[j] > t; j--)
```

```
                    data[j + 1] = data[j];
                data[j] = t;
        }
}
```

PASCAL

程式使用 linked list 做插入排序，目的：将读入的英文名字按字母排列

```
TYPE
link=^node;
node=record
       data:string;
       next:link;
     end;
VAR

p,q,head,n:link;
t,m:integer;
f1,f2:text;
i:string;
BEGIN

assign(f1,'lianbiao-name-in.txt');
reset(f1);
assign(f2,'lianbiao-name-out.txt');
rewrite(f2);
head:=nil;
read(f1,t);
readln(f1);
read(f1,i);
new(p);
p^.data:=i;
p^.next:=nil;
head:=p;
readln(f1);
read(f1,i);
FOR m:=2 TO t DO
 BEGIN
  p:=head;
```

```
  new(n);
  n^.data:=i;
  while (i>p^.data) and (p^.next<>nil) do
   begin
    q:=p;
    p:=p^.next;
   end;
  if i<head^.data then begin
                         n^.next:=head;
                         head:=n;
                       end
                  else if (i>p^.data) and (p^.next=nil) then begin
                                                               p^.next:=n;
                                                               n^.next:=nil;
                                                             end
                                                        else begin
                                                               q^.next:=n;
                                                               n^.next:=p;
                                                             end;
  readln(f1);
  read(f1,i);
 end;
p:=head;
while p<>nil do
 begin
  write(f2,p^.data,' ');
  p:=p^.next;
 end;
CLOSE(f1);
CLOSE(f2);
END.
```

Python

```
def insertion_sort(n):
    if len(n) == 1:
        return n
    b = insertion_sort(n[1:])
    m = len(b)
```

```
    for i in range(m):
        if n[0] <= b[i]:
            return b[:i]+[n[0]]+b[i:]
    return b + [n[0]]
```

Java

```
        public static void insertion_sort( int[] arr ) {
                for( int i=0; i<arr.length-1; i++ ) {
                        for( int j=i+1; j>0; j-- ) {
                                if( arr[j-1] > arr[j] ) {
                                        int temp = arr[j];
                                        arr[j] = arr[j-1];
                                        arr[j-1] = temp;
                                } else {
                                        break;
                                }
                        }
                }
        }
```

JavaScript

```
Array.prototype.insertion_sort = function() {
        var i, j;
        var temp;
        for (i = 1; i < this.length; i++) {
                temp = this[i];
                for (j = i - 1; j >= 0 && this[j] > temp; j--)
                        this[j + 1] = this[j];
                this[j + 1] = temp;
        }
        return this;
};
用法示例：[3,5,2,11,1,2,”abc","zfd","sad","eng"].insert_sort();
```

PHP

```php
function insertion_sort(&$arr) { //php的陣列視為基本型別，所以必須用傳參考才能修改原陣列
        for ($i = 1; $i < count($arr); $i++) {
                $temp = $arr[$i];
                for ($j = $i - 1; $j >= 0 && $arr[$j] > $temp; $j--)
                        $arr[$j + 1] = $arr[$j];
                $arr[$j + 1] = $temp;
        }
}
```

算法复杂度

如果目标是把n个元素的序列升序排列，那么采用**插入排序**存在最好情况和最坏情况。最好情况就是，序列已经是升序排列了，在这种情况下，需要进行的比较操作需(n−1)次即可。最坏情况就是，序列是降序排列，那么此时需要进行的比较共有n(n−1)/2次。**插入排序**的赋值操作是比较操作的次数减去(n−1)次。平均来说**插入排序**算法复杂度为**O(n²)**。因而，**插入排序**不适合对于数据量比较大的排序应用。但是，如果需要排序的数据量很小，例如，量级小于千，那么**插入排序**还是一个不错的选择。 插入排序在工业级库中也有着广泛的应用，在STL的sort算法和stdlib的qsort算法中，都将插入排序作为快速排序的补充，用于少量元素的排序（通常为8个或以下）。