Implementation

Language: C

```
int* sort (int* A, int n)
{
        // index i divide A[] to 2 partitions
        // A[0…i-1] are sorted, A[i…n-1] are unsorted
        for (int i = 1; i < n; ++i)
        {
                int j = i, val = A[j];
                while (j > 0 && !(A[j-1] <= val))
                {
                        A[j] = A[j-1];
                        --j;
                }
                A[j] = val;
        }
        return A;
}
```

n=4

Best

index: 0 1 2 3

A=[1,2,3,4]

j=1, val=2

j=2, val=3

j=3, val=4

A[1]=2 → C_1

A[2]=3 → C_2

A[3]=4 → C_3

n=4

Worst

index: 0 1 2 3

A=[4,3,2,1]

i=1, j=1, val=3

A[1]=4

A[0]=3 → 1次

A=[3,4,2,1]

i=2, j=2, val=2

A[2]=4

j=1

A[1]=3 → 2次

A[0]=2

A=[2,3,4,1]

i=3, j=3, val=1

A[3]=4

j=2

A[2]=3

j=1

A[1]=2

A[0]=1 → 3次

Analysis

Space complexity

Assume that instance is an n elements array.

- Instance: n values and 1 index (A, n)
- Divider: 1 index (i)
- Ordering maintain: 1 index (j) and 1 value(val)

Totally needs $n+1$ values and 3 indices, so the space complexity is $O(n)$

Time complexity

Best case: 資料已完成排序的情況下，只需執行最外層的 for loop n-1 次

e.g. A=[1,2,3,4] 要執行 3 次，$T(4) = C_1+C_2+C_3$，$O(n-1) = O(n) = n$

(因為 A[j-1] 會永遠 < val，所以不會進去 while loop)

Worst case: 資料剛好是由小到大排序的時候，每取一個新元素都要將元素插入到序列最前面

e.g. A=[4,3,2,1] 要執行 6 次，$T(4) = 1+2+3 = 6$

$1+2+3+\cdots+(n-1) = \frac{n\cdot(n-1)}{2} = \frac{n^2}{2} - \frac{n}{2}$

$O(\frac{n^2}{2} - \frac{n}{2}) = n^2$ (捨去低次項和係數)