选择排序

最差时间复杂度
$O(n^2)$
最优时间复杂度
$O(n^2)$
平均时间复杂度
$O(n^2)$
最差空间复杂度
O(n) total, O(1) auxiliary
选择排序（Selection sort）是一种简单直观的排序算法。它的工作原理如下。首先在未排序序列中找到最小（大）元素，存放到排序序列的起始位置，然后，再从剩余未排序元素中继续寻找最小（大）元素，然后放到已排序序列的末尾。以此类推，直到所有元素均排序完毕。
选择排序的主要优点与数据移动有关。如果某个元素位于正确的最终位置上，则它不会被移动。选择排序每次交换一对元素，它们当中至少有一个将被移到其最终位置上，因此对n个元素的表进行排序总共进行至多n−1次交换。在所有的完全依靠交换去移动元素的排序方法中，选择排序属于非常好的一种。

实作范例

C语言

```c
void selection_sort(int arr[], int len) {
        int i, j, min, temp;
        for (i = 0; i < len - 1; i++) {
                min = i;
                for (j = i + 1; j < len; j++)
                        if (arr[min] > arr[j])
                                min = j;
                temp = arr[min];
                arr[min] = arr[i];
                arr[i] = temp;
        }
}
```

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 selection_sort(T arr[], int len)
{
  int i, j, min;
  for (i = 0; i < len - 1; i++)
  {
     min = i;
     for (j = i + 1; j < len; j++)
        if(arr[min] > arr[j])
              min = j;
     swap(arr[i], arr[min]);
  }
}
```

C#

```
static void selection_sort<T>(T[] arr) where T : System.IComparable<T>{//整數或浮點數皆可使用
        int i, j, min, len = arr.Length;
        T temp;
        for (i = 0; i < len - 1; i++) {
                min = i;
                for (j = i + 1; j < len; j++)
                        if (arr[min].CompareTo(arr[j]) > 0)
                                min = j;
                temp = arr[min];
                arr[min] = arr[i];
                arr[i] = temp;
        }
}
```

VB.net

```
'進行排序前先建構兩數值交換的程式switch
   Private Sub switch(ByRef a, ByRef b)
      Dim c As Integer
```

```
    c = a: a = b: b = c
End Sub

'選擇排序由小到大
    Dim i, j, count As Integer
    For i = 0 To UBound(b) – 2
        For j = i + 1 To UBound(b)–1
            If b(i) > b(j) Then
                switch(b(i), b(j))
                count += 1
            end if
        Next
    Next

    MsgBox("一共經過了" & count & “次的排序")
```

Python

```
def selection_sort(L):
    N = len(L)
    exchanges_count = 0
    for i in range(N–1):
        min_index = i
        for j in range(i+1, N):
            if L[min_index] > L[j]:
                min_index = j
        if min_index != i:
            L[min_index], L[i] = L[i], L[min_index]
            exchanges_count += 1
        print('iteration #{}: {}'.format(i, L))
    print('Total {} swappings'.format(exchanges_count))
    return L

testlist = [17, 23, 20, 14, 12, 25, 1, 20, 81, 14, 11, 12]
print('Before selection sort: {}'.format(testlist))
print('After selection sort:  {}’.format(selection_sort(testlist)))
```

Java

```
public static void selection_sort(int[] arr) {
	int i, j, min, temp, len = arr.length;
	for (i = 0; i < len - 1; i++) {
		min = i;
		for (j = i + 1; j < len; j++)
			if (arr[min] > arr[j])
				min = j;
		temp = arr[min];
		arr[min] = arr[i];
		arr[i] = temp;
	}
}
```

JavaScript

```
Array.prototype.selection_sort = function() {
	var i, j, min;
	var temp;
	for (i = 0; i < this.length - 1; i++) {
		min = i;
		for (j = i + 1; j < this.length; j++)
			if (this[min] > this[j])
				min = j;
		temp = this[min];
		this[min] = this[i];
		this[i] = temp;
	}
};
```

PHP

```
function swap(&$x, &$y) {
	$t = $x;
	$x = $y;
	$y = $t;
}
function selection_sort(&$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 = 0; $i < count($arr) - 1; $i++) {
```

```
            $min = $i;
            for ($j = $i + 1; $j < count($arr); $j++)
                if ($arr[$min] > $arr[$j])
                    $min = $j;
            swap($arr[$min], $arr[$i]);
        }
}
```

复杂度分析

选择排序的**交换操作**介于0和$(n-1)$次之间。选择排序的**比较操作**为$n(n-1)/2$次之间。选择排序的**赋值操作**介于0和$3(n-1)$次之间。

比较次数$O(n^2)$，比较次数与关键字的初始状态无关，总的比较次数$N=(n-1)+(n-2)+...+1=n\times(n-1)/2$。交换次数$O(n)$，最好情况是，已经有序，交换0次；最坏情况是，逆序，交换$n-1$次。交换次数比冒泡排序较少，由于交换所需CPU时间比比较所需的CPU时间多，n值较小时，选择排序比冒泡排序快。
原地操作几乎是选择排序的唯一优点，当方度（space complexity）要求较高时，可以考虑选择排序；实际适用的场合非常罕见。