C 語 言 專 題
不 同 排 序 法 比 較

問題： 輸入一個整數陣列 ，將陣列元素由小排到大

方法一： 泡沫排序(Bubble Sort)，一層for將第 j 與 j+1 個值做比較，若第 j 個值比第 j+1 個值大，則位置交換。這層for run完後就決定了最大一數在最右邊，再用一層for讓每個值都執行此動作，但比到第 n-i 位前即可。

實際程式碼：

```c
#include<stdio.h>
//bubble
int main(){
    int i,j,temp = 0;
    int n;
    printf("n =");
    scanf("%d",&n);
    int a[n];
    printf("intput numbers:");
    for(i = 0; i < n; i++){
        scanf("%d",&a[i]);
    }
    for(i = 0; i < n; i++){                    //兩個for : 一個for只能做到交換兩數，e
        for(j = 0; j < n - i; j++){            //[5,4,3,2,1]  when i = 0; [4,3,2,1,5]
            if(a[j] > a[j + 1]){                                    // i = 1; [3,2,1,4,5]
                temp = a[j];                                        // i = 2; [2,1,3,4,5]
                a[j] = a[j + 1];                                    // i = 3; [1,2,3,4,5]
                a[j + 1] = temp;
            }
        }
    }
    for(i  = 0; i < n; i++){
        printf("%d ",a[i]);
    }
    return 0;
}
```

時間複雜度： Best Case：O(n)
Worst Case：O(n^2)
Average Case：O(n^2)

空間複雜度： S(n) = O(1)

方法二：選擇排序(Selection Sort)，第一個for從陣列中找出最小值，並與第一個值做交換，再用一層for依序將所有位置的值確定，簡單來說就是不斷找出最小值並將他們交換到正確位置。也有寫法是給定兩陣列，分別為未排序及已排序，在未排序陣列中不斷找出最小值，再丟到已排序陣列的最右邊(或丟到最左邊，依序往右排)。

實際程式碼：

```c
#include<stdio.h>

int main(){
  int n, i,j,min = 0;
  int minindex = 0; //record the min index
  printf("n = ");
  scanf("%d", &n);
  int a[n];
  printf("input numbers : ");
  for(i = 0; i < n; i++){
    scanf("%d", &a[i]);
  }
    for(i = 0; i < n-1; i++){
        int min = 2147483647;
        for(j = i+1; j < n; j++){
            if (a[j] < min){
                min = a[j];           //找出最小值
                minindex = j;
                printf("min = %d\n",min);
            }
        }
        if(min < a[i])
        {
            int temp;             //將未排序的最左邊數與最小值交換
            temp = a[i] ; // temp = 最左邊未排序的數
            a[i] = a[minindex]; // 最左邊未排序的數 = 未排序之中最小值
            a[minindex] = temp;//最小值的位置 = 最左邊未排序的數
        }
    }
  for(i = 0; i < n; i++){
    printf("%d ", a[i]);
  }
  return 0;
}
```

時間複雜度： Average/Worst/Best Case 皆為 O(n^2)

空間複雜度： S(n) = O(1)

方法三：插入排序(Insertion Sort)，跟選擇排序有點像，不同的點在於選擇排序是不斷找出最小值依序往前交換；插入排序是依序將數一個一個插入對的位置。

實際程式碼：

```c
#include<stdio.h>
//插入排序
int main(){
    int n;
    printf("n = ");
    scanf("%d",&n);
    int a[n];
    printf("input numbers : ");
    for(int i = 0; i < n; i++){
        scanf("%d",&a[i]);
    }
    for(int i = 0; i < n; i++){
        int j = i;                                  //依序插入
        while(j > 0 && a[j - 1] > a[j]){            //若小於前一個數，則交換
            int temp = a[j];
            a[j] = a[j - 1];
            a[j - 1] = temp;
            j--;                                    //因為由尾部開始做
        }
    }
    for(int i = 0; i < n; i++){
        printf("%d ",a[i]);
    }
return 0;
}
```

時間複雜度： Best Case：O(1)
Worst Case：O(n^2)
Average Case：O(n^2)

空間複雜度： S(n) = O(1)

方法四： 快速排序(Quick Sort)，先找一個基準點(pivot)，然後派right及left分別從資料的兩邊開始往中間找，如果右邊找到一個值比基準點小，左邊找到一個值比基準點大，就讓他們互換。反覆找並互換，直到相遇。然後再將相遇的點跟pivot互換。再用一個迴圈重複此動作，直到排序完成。

實際程式碼：

```c
#include <stdio.h>
int a[100],n;

void quicksort ( int left, int right ){
    int i, j, t, pivot;
    if (left > right){
        return ;
    }
    pivot = a[left];      //最左側為pivot
    i = left;
    j = right;
    while (i != j){
        while (a[j] >= pivot && i < j)       //找比pivot小的值
            j--;
        while (a[i] <= pivot && i < j)       //找比pivot大的值
            i++;
        if (i < j){          //left還沒遇到right
            t = a[i];
            a[i] = a[j];
            a[j] = t;
        }
    }
    a[left] = a[i];      //left與right相遇後，與pivot交換
    a[i] = pivot;
    quicksort (left, i - 1);          //呼叫遞迴一直做到排序完成
    quicksort (i + 1, right);
}

int main(){
    int i, j, t;
    printf("n = ");
    scanf ("%d", &n);
    printf("input numbers : ");
    for (i = 1; i <= n; i++){
        scanf ("%d", &a[i]);
    }
    quicksort ( 1, n );
    for (i = 1; i <= n; i++){
        printf ("%d ", a[i]) ;
```

時間複雜度： Best Case：O(n log n)
Worst Case：O(n^2)
Average Case：O(n log n)

空間複雜度： S(n) = O(log n) ~ O(n)

方法五：合併排序(Merge Sort)，與quick sort一樣是divide and conquer的概念，做法為將陣列複製到子陣列，直到成為一個一個單獨，再兩兩比較後依序插入新的子陣列。

實際程式碼：

```c
#include <stdio.h>
#include <stdlib.h>
// First subarray is arr[head..mid]
// Second subarray is arr[mid+1..tail]
void merge(int arr[], int head, int mid, int tail){
    int lenA = mid - head + 1;          //一陣列分為前後兩段
    int lenB = tail - (mid + 1) + 1;
    int A[lenA];
    int B[lenB];
    int i, j, k;
    for(i = 0; i < lenA; i++){   //將陣列複製到A、B子陣列
        A[i] = arr[head + i];
    }
    for(j = 0; j < lenB; j++){
        B[j] = arr[mid + 1 + j];
    }
    i = 0;
    j = 0;
    k = head;
    while(i < lenA && j < lenB){      //當陣列還沒掃完
        if(A[i] < B[j]){     //兩兩比大小，比完後小的先排進子陣列
            arr[k] = A[i];
            i++;
        }else{
            arr[k] = B[j];
            j++;
        }
        k++;
    }
    while(i < lenA){     //複製"經一輪排序後"的陣列到另一子陣列
        arr[k] = A[i];
        i++;
        k++;
    }
    while(j < lenB){
        arr[k] = B[j];
        j++;
        k++;
    }
}
void merge_sort(int arr[], int head, int tail){
    if(head < tail){                    //重複做到每個單位獨立成一個，也就是head = tail
        int mid = (head + tail) / 2;     //每次都對分
        merge_sort(arr, head, mid);      //分兩段
        merge_sort(arr, mid+1, tail);
        merge(arr, head, mid, tail);
    }
```

時間複雜度： Average/Best/Worst Case 的時間皆為 O(n log n)

空間複雜度： S(n) = O(n)

不同排序法比較-結論

1. 此五種排序法中，泡沫排序最爲直觀，而選擇排序與插入排序在生活上很常使用，例如：要將撲克牌照大小排時就會使用此方法。
2. 雖前三種的平均時間複雜度皆爲O(n^2)，但其中的插入排序在Best Case時可達到O(1)，非常快速。
若能將此方法結合快速排序或合併排序中的Divide and Conquer的概念，將大問題先切分成小問題，便能做出更有效率的排序法。
3. 快速排序非常看重每次pivot的選擇，這次我寫的只是基本選最左邊的當pivot，若能用簡單的機制，如：選其中三數的中位數當pivot，便能避免Worst Case的發生。
4. 合併排序概念上很有趣，但在呼叫遞迴式的地方是比較難理解的。雖在複製與合併陣列的過程使用了較多的陣列空間，但其時間複雜度相對穩定。

thanks for reading