109-1 4101031_01 程式設計 (一) HW5　　資工 2B　　408410120　　鍾博丞

本次實驗測試環境：

OS: Ubuntu 18.04　　CPU: AMD R9 3900X @4.2 GHz　　RAM: 32GB 3600MHz CL16

MB: ASUS TUF-X570　　Coding environment: Visual Studio Code

A. 測試排序的效能

本次實驗建立 10 萬筆整數資料進行排序，分別有呼叫 rand()產生亂數的陣列排序(Average case)，與反序排列的陣列排序(Worst case)

排序方式分別有 Bubble sort, Selection sort, Insertion sort, Heap sort, Quick sort.

實驗結果

100K random numbers					
Algorithm	Bubble	Selection	Insertion	Heap	Quick
1	25.662052	28.844402	4.527063	0.018164	0.010785
2	25.830038	27.668844	4.404304	0.018178	0.01075
3	26.746975	28.077617	4.7375	0.018218	0.010767
4	26.505158	27.417324	4.729528	0.018158	0.010853
5	26.626215	27.364085	4.365437	0.018154	0.011094
6	24.520053	27.022386	4.494025	0.017504	0.010478
7	26.849636	27.365421	4.379605	0.018208	0.010848
8	26.579461	27.301448	4.353858	0.018117	0.010672
9	27.037114	27.26036	4.644524	0.018007	0.010813
10	25.527386	26.577741	4.548214	0.017569	0.010681
Average	26.1884088	27.4899628	4.5184058	0.0180277	0.0107741
Standard deviation	0.709691841	0.554936992	0.132390218	0.000240433	0.000142399

100K reverse order numbers					
Algorithm	Bubble	Selection	Insertion	Heap	Quick
1	18.560646	18.247368	8.883443	0.013961	8.233166

排序的方法詳見完整程式碼

Bubble, selection, insertion sort 的時間複雜度都是 $O(n^2)$，但是 insertion sort 的效能明顯比其他兩個快很多，判斷應該是因為 insertion sort 不是利用 swap，而是用平移陣列的方式進行所造成的現象。

B. 測試排序的穩定性

我們將初始陣列設定值為{ 3,5,3,1,2,5,4,1,2,4 }，並多設一個陣列位置
{ 0,1,2,3,4,5,6,7,8,9 }，每次排序時，有呼叫 swap()時，也 swap 陣列位置，藉以判斷其相同數值之位置有沒有被交換

實驗結果

The order after bubble sort:

1 1 2 2 3 3 4 4 5 5

3 7 4 8 0 2 6 9 1 5

The order after heap sort:

1 1 2 2 3 3 4 4 5 5

3 7 **8 4 2 0** 6 9 **5 1**

The order after selection sort:

1 1 2 2 3 3 4 4 5 5

3 7 4 8 **2 0** 6 9 **5 1**

The order after quick sort:

1 1 2 2 3 3 4 4 5 5

7 3 8 4 2 0 6 9 1 5

The order after insertion sort:

1 1 2 2 3 3 4 4 5 5

3 7 4 8 0 2 6 9 1 5

1. Bubble sort:

Bubble sort 的概念就是一直把大數字往後移，比較的是相鄰的兩數字，如果兩數字相等，則不會交換；就算透過前面的交換使得這兩個數值相鄰，他們也不會交換。

故而：Bubble sort 是**穩定**的排序演算法

2. Selection sort

Selection sort 的概念是找出第一個數字之後的最小數值，與第一個數字比較，若該最小數值比較小，則交換，然後找出第二個數字之後的最小數值，與第二個數字比較，以此類推。這會造成一個現象，如果兩相同數字 (不論是否相鄰) 的後面有數字比他們小，則第一個數字就會與這個小數字交換，那麼這兩個相同數字的位置也交換了

例如本次實驗：3, 5, 3, 1, …，第一個 3 會與 1 交換，那麼兩個 3 的順序也被交換了

故而：Selection sort 是**不穩定**的排序演算法

3. Insertion sort

Insertion sort 的概念是在一個已經排序完成的小序列上，插入一個數值，而此數值會從這個小序列的最後一個開始掃描，若此數值大於或**等於**小序列的的某一值，則會將此數值插在此值的**後面**，所以，相等的元素順序是不會改變的

故而：Insertion sort 是**穩定**的排序演算法

附錄：程式碼

HW5_brenchmark.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <sys/time.h>
#define TEST_DATA_CNT 100000

void swap(int* x, int* y) {
    int temp = *x;
    *x = *y;
    *y = temp;
}

void bubblesort(int* A, int length) {
    int i, j, temp;
    for (i = length - 1; i > 0; --i) {
        for (j = 0; j < i; ++j) {
            if (A[j] > A[j + 1]) {
                swap(&A[j], &A[j + 1]);
            }
        }
    }
}

void selectionsort(int* A, int length) {
    int i, j, temp, min;
    for (i = 0; i < length - 1; ++i) {
        min = i;
        for (j = i + 1; j < length; ++j) {
            if (A[min] > A[j]) {
                min = j;
            }
        }
        swap(&A[i], &A[min]);
    }
}

void insertionsort(int* A, int length) {
    int i, j, temp, key;
    for (int i = 1; i < length; i++) {
        int key = A[i];
        int j = i - 1;
        while (key < A[j] && j >= 0) {
            A[j + 1] = A[j];
            --j;
        }
        A[j + 1] = key;
    }
}

void shiftdown(int* A, int size, int index) {
    int left = index * 2 + 1;
    int right = left + 1;
    int max = left;
    int temp;
    if (right == size) {
        if (A[index] < A[left])
            swap(&A[index], &A[left]);
    }
    else if (right < size) {
        if (A[left] < A[right])
            max = right;
        if (A[index] < A[max]) {
            swap(&A[index], &A[max]);
            shiftdown(A, size, max);
```

```
        }
    }
}

void heapsort(int* A, int size) {
    int i, temp;
    for (i = size - 1; i >= 0; --i)
        shiftdown(A, size, i);

    while (size > 1) {
        swap(&A[0], &A[size - 1]);
        --size;
        shiftdown(A, size, 0);
    }
}

int partition(int* A, int p, int r) {
    int i = p + 1, j = r, temp;
    do {
        while (A[i] <= A[p]) {
            if (i == r)
                break;
            else
                ++i;
        }
        while (A[j] >= A[p]) {
            if (j == p)
                break;
            else
                --j;
        }
        if (i < j) {
            swap(&A[i], &A[j]);
        }
    } while (i < j);
    if (j != p)
        swap(&A[p], &A[j]);
    return j;
}

void quicksort(int* A, int p, int r) {
    if (p < r) {
        int q = partition(A, p, r);
        quicksort(A, p, q - 1);
        quicksort(A, q + 1, r);
    }
}

int main() {
    int i;
    int test_data[TEST_DATA_CNT + 5];
    struct timeval start;
    struct timeval end;
    unsigned long diff;

    printf("There are %d numbers in array.\n", TEST_DATA_CNT);

    /*********************************/
    /*      Create Random Data       */
    /*********************************/

    srand(time(NULL));

    /************************************************/
    /*      Test Performance for random number      */
    /************************************************/
    printf("Test Performance for random number\n");

    for (i = 0; i < TEST_DATA_CNT; ++i) {
```

```
        test_data[i] = rand();
    }

    gettimeofday(&start, NULL);
    bubblesort(test_data, TEST_DATA_CNT);
    gettimeofday(&end, NULL);
    diff = 1000000 * (end.tv_sec - start.tv_sec) + end.tv_usec - start.tv_usec;     // 實際的時間差
    printf("The performance of bubble sort is %ld us (equal %f sec)\n", diff, diff / 1000000.0);

    for (i = 0; i < TEST_DATA_CNT; ++i) {
        test_data[i] = rand();
    }

    gettimeofday(&start, NULL);
    selectionsort(test_data, TEST_DATA_CNT);
    gettimeofday(&end, NULL);
    diff = 1000000 * (end.tv_sec - start.tv_sec) + end.tv_usec - start.tv_usec;     // 實際的時間差
    printf("The performance of selection sort is %ld us (equal %f sec)\n", diff, diff / 1000000.0);

    for (i = 0; i < TEST_DATA_CNT; ++i) {
        test_data[i] = rand();
    }

    gettimeofday(&start, NULL);
    insertionsort(test_data, TEST_DATA_CNT);
    gettimeofday(&end, NULL);
    diff = 1000000 * (end.tv_sec - start.tv_sec) + end.tv_usec - start.tv_usec;     // 實際的時間差
    printf("The performance of insertion sort is %ld us (equal %f sec)\n", diff, diff / 1000000.0);

    for (i = 0; i < TEST_DATA_CNT; ++i) {
        test_data[i] = rand();
    }

    gettimeofday(&start, NULL);
    heapsort(test_data, TEST_DATA_CNT);
    gettimeofday(&end, NULL);
    diff = 1000000 * (end.tv_sec - start.tv_sec) + end.tv_usec - start.tv_usec;     // 實際的時間差
    printf("The performance of heap sort is %ld us (equal %f sec)\n", diff, diff / 1000000.0);

    for (i = 0; i < TEST_DATA_CNT; ++i) {
        test_data[i] = rand();
    }

    gettimeofday(&start, NULL);
    quicksort(test_data, 0, TEST_DATA_CNT - 1);
    gettimeofday(&end, NULL);
    diff = 1000000 * (end.tv_sec - start.tv_sec) + end.tv_usec - start.tv_usec;     // 實際的時間差
    printf("The performance of quick sort is %ld us (equal %f sec)\n", diff, diff / 1000000.0);



////////////////////////////////////////////////////////////////////////////////////////////////////


    /*************************************************/
    /*      Test Performance for reverse number      */
    /*************************************************/
    printf("Test Performance for reverse number\n");

    for (i = 0; i < TEST_DATA_CNT; ++i) {
        test_data[i] = TEST_DATA_CNT - i;
    }

    gettimeofday(&start, NULL);
    bubblesort(test_data, TEST_DATA_CNT);
    gettimeofday(&end, NULL);
    diff = 1000000 * (end.tv_sec - start.tv_sec) + end.tv_usec - start.tv_usec;
    printf("The performance of bubble sort is %ld us (equal %f sec)\n", diff, diff / 1000000.0);
```

```
    for (i = 0; i < TEST_DATA_CNT; ++i) {
        test_data[i] = TEST_DATA_CNT - i;
    }

    gettimeofday(&start, NULL);
    selectionsort(test_data, TEST_DATA_CNT);
    gettimeofday(&end, NULL);
    diff = 1000000 * (end.tv_sec - start.tv_sec) + end.tv_usec - start.tv_usec;
    printf("The performance of selection sort is %ld us (equal %f sec)\n", diff, diff / 1000000.0);

    for (i = 0; i < TEST_DATA_CNT; ++i) {
        test_data[i] = TEST_DATA_CNT - i;
    }

    gettimeofday(&start, NULL);
    insertionsort(test_data, TEST_DATA_CNT);
    gettimeofday(&end, NULL);
    diff = 1000000 * (end.tv_sec - start.tv_sec) + end.tv_usec - start.tv_usec;
    printf("The performance of insertion sort is %ld us (equal %f sec)\n", diff, diff / 1000000.0);

    for (i = 0; i < TEST_DATA_CNT; ++i) {
        test_data[i] = TEST_DATA_CNT - i;
    }

    gettimeofday(&start, NULL);
    heapsort(test_data, TEST_DATA_CNT);
    gettimeofday(&end, NULL);
    diff = 1000000 * (end.tv_sec - start.tv_sec) + end.tv_usec - start.tv_usec;
    printf("The performance of heap sort is %ld us (equal %f sec)\n", diff, diff / 1000000.0);

    for (i = 0; i < TEST_DATA_CNT; ++i) {
        test_data[i] = TEST_DATA_CNT - i;
    }

    gettimeofday(&start, NULL);
    quicksort(test_data, 0, TEST_DATA_CNT - 1);
    gettimeofday(&end, NULL);
    diff = 1000000 * (end.tv_sec - start.tv_sec) + end.tv_usec - start.tv_usec;
    printf("The performance of quick sort is %ld us (equal %f sec)\n", diff, diff / 1000000.0);
    return 0;
}
```

HW5_stability test.c

```c
#include <stdio.h>
#include <stdlib.h>
#define TEST_DATA_CNT 10

void swap(int* x, int* y) {
    int temp = *x;
    *x = *y;
    *y = temp;
}

void bubblesort(int* A, int* B, int length) {
    int i, j, temp;
    for (i = length - 1; i > 0; --i) {
        for (j = 0; j < i; ++j) {
            if (A[j] > A[j + 1]) {
                swap(&A[j], &A[j + 1]);
                swap(&B[j], &B[j + 1]);
            }
        }
    }
}

void selectionsort(int* A, int* B, int length) {
    int i, j, temp, min;
    for (i = 0; i < length - 1; ++i) {
        min = i;
        for (j = i + 1; j < length; ++j) {
            if (A[min] > A[j]) {
                min = j;
            }
        }
        swap(&A[i], &A[min]);
        swap(&B[i], &B[min]);
    }
}

void insertionsort(int* A, int* B, int length) {
    int i, j, temp, key, _key;
    for (int i = 1; i < length; i++) {
        int key = A[i];
        int _key = B[i];
        int j = i - 1;
        while (key < A[j] && j >= 0) {
            A[j + 1] = A[j];
            B[j + 1] = B[j];
            --j;
        }
        A[j + 1] = key;
        B[j + 1] = _key;
    }
}

void shiftdown(int* A, int* B, int size, int index) {
    int left = index * 2 + 1;
    int right = left + 1;
    int max = left;
    int temp;
    if (right == size) {
        if (A[index] < A[left]) {
            swap(&A[index], &A[left]);
            swap(&B[index], &B[left]);
        }
    }
    else if (right < size) {
        if (A[left] < A[right])
            max = right;
        if (A[index] < A[max]) {
```

```
            swap(&A[index], &A[max]);
            swap(&B[index], &B[max]);
            shiftdown(A, B, size, max);
        }
    }
}

void heapsort(int* A, int* B, int size) {
    int i, temp;
    for (i = size - 1; i >= 0; --i)
        shiftdown(A, B, size, i);

    while (size > 1) {
        swap(&A[0], &A[size - 1]);
        swap(&B[0], &B[size - 1]);
        --size;
        shiftdown(A, B, size, 0);
    }
}

int partition(int* A, int* B, int p, int r) {
    int i = p + 1, j = r, temp;
    do {
        while (A[i] <= A[p]) {
            if (i == r)
                break;
            else
                ++i;
        }
        while (A[j] >= A[p]) {
            if (j == p) {
                break;
            else
                --j;
        }
        if (i < j) {
            swap(&A[i], &A[j]);
            swap(&B[i], &B[j]);
        }
    } while (i < j);
    if (j != p) {
        swap(&A[p], &A[j]);
        swap(&B[p], &B[j]);
    }
    return j;
}

void quicksort(int* A, int* B, int p, int r) {
    if (p < r) {
        int q = partition(A, B, p, r);
        quicksort(A, B, p, q - 1);
        quicksort(A, B, q + 1, r);
    }
}

int main() {
    int i, j;
    const int test_data[TEST_DATA_CNT] = { 3,5,3,1,2,5,4,1,2,4 };
    const int test_order[TEST_DATA_CNT] = { 0,1,2,3,4,5,6,7,8,9 };
    int execute_test_data[TEST_DATA_CNT] = { 3,5,3,1,2,5,4,1,2,4 };
    int execute_test_order[TEST_DATA_CNT] = { 0,1,2,3,4,5,6,7,8,9 };

    bubblesort(execute_test_data, execute_test_order, TEST_DATA_CNT);
    printf("The order after bubble sort:\n");
    for (i = 0; i < TEST_DATA_CNT; ++i) {
        printf("%d ", execute_test_data[i]);
    }
    putchar('\n');
    for (i = 0; i < TEST_DATA_CNT; ++i) {
```

```
        printf("%d ", execute_test_order[i]);
    }
    putchar('\n');
    for (i = 0; i < TEST_DATA_CNT; ++i) {
        execute_test_data[i] = test_data[i];
        execute_test_order[i] = test_order[i];
    }
    selectionsort(execute_test_data, execute_test_order, TEST_DATA_CNT);
    printf("The order after selection sort:\n");
    for (i = 0; i < TEST_DATA_CNT; ++i) {
        printf("%d ", execute_test_data[i]);
    }
    putchar('\n');
    for (i = 0; i < TEST_DATA_CNT; ++i) {
        printf("%d ", execute_test_order[i]);
    }
    putchar('\n');
    for (i = 0; i < TEST_DATA_CNT; ++i) {
        execute_test_data[i] = test_data[i];
        execute_test_order[i] = test_order[i];
    }
    insertionsort(execute_test_data, execute_test_order, TEST_DATA_CNT);
    printf("The order after insertion sort:\n");
    for (i = 0; i < TEST_DATA_CNT; ++i) {
        printf("%d ", execute_test_data[i]);
    }
    putchar('\n');
    for (i = 0; i < TEST_DATA_CNT; ++i) {
        printf("%d ", execute_test_order[i]);
    }
    putchar('\n');
    for (i = 0; i < TEST_DATA_CNT; ++i) {
        execute_test_data[i] = test_data[i];
        execute_test_order[i] = test_order[i];
    }
    heapsort(execute_test_data, execute_test_order, TEST_DATA_CNT);
    printf("The order after heap sort:\n");
    for (i = 0; i < TEST_DATA_CNT; ++i) {
        printf("%d ", execute_test_data[i]);
    }
    putchar('\n');
    for (i = 0; i < TEST_DATA_CNT; ++i) {
        printf("%d ", execute_test_order[i]);
    }
    putchar('\n');
    for (i = 0; i < TEST_DATA_CNT; ++i) {
        execute_test_data[i] = test_data[i];
        execute_test_order[i] = test_order[i];
    }
    quicksort(execute_test_data, execute_test_order, 0, TEST_DATA_CNT - 1);
    printf("The order after quick sort:\n");
    for (i = 0; i < TEST_DATA_CNT; ++i) {
        printf("%d ", execute_test_data[i]);
    }
    putchar('\n');
    for (i = 0; i < TEST_DATA_CNT; ++i) {
        printf("%d ", execute_test_order[i]);
    }
    putchar('\n');
    for (i = 0; i < TEST_DATA_CNT; ++i) {
        execute_test_data[i] = test_data[i];
        execute_test_order[i] = test_order[i];
    }

    return 0;
}
```