

分析 Sorting 時間

計算機演算法 作業 1

姓名：巫逸哲

班級：資訊三乙

學號：D0641604

目錄

壹、	規則建立說明	1
貳、	程式碼	1
參、	執行結果	12
肆、	討論	13
伍、	心得	13

計 算 機 演 算 法 作 業 1 分 析 S o r t i n g 時 間

壹、 規則建立說明

分別建立五種排序法的程式，執行時會計算執行時間，並分別輸出執行結

果，並建立一個批次檔輪流執行五種排序法。

貳、 程式碼

一、氣泡排序 (Bubble sort)

```
# include <stdio.h>
# include <stdlib.h>
# include <unistd.h>
# include <windows.h>
# include <time.h>
# define TIMES 25
# define MAX 50000

int *generate_number (int *data) {
    for (int i = 0; i < MAX; i++) {
        data[i] = rand() % 1000000 + 1;
        ((i+1)%10 == 0) ? printf("%10d\n", data[i]) : printf("
%10d ", data[i]);
    }
    return (int *) data;
}

void check_number (int *data) {
    for (int i = 1; i < MAX; i++) {
        ((i+1)%10 == 0) ? printf("%10d\n", data[i]) : printf("
%10d ", data[i]);
    }
}

double bubble_sort (int *data) {
    srand(time(NULL));
```

```

        clock_t t_start = clock();

        for (int i = 0; i < MAX; i++) {
            for (int j = 0; j < MAX-i-1; j++) {
                if (data[j] > data[j+1]) {
                    int temp = data[j];
                    data[j] = data[j+1];
                    data[j+1] = temp;
                }
            }
        }

        clock_t t_end = clock();
        return ((double)t_end - t_start) / CLOCKS_PER_SEC;
    }

int main () {
    int data[MAX] = {0};
    double times[TIMES] = {0};

    FILE *output = fopen("bubble_sort_output.txt", "w");

    for (int i = 0; i < TIMES; i++) {
        generate_number(data);
        printf("\n\t%2d times Sorting Start...\n\n\n", i + 1);

        times[i] = bubble_sort(data);
        sleep(1);

        printf("\n\t%2d times Sorting End...\n\n\n", i + 1);
        check_number(data);

        printf("\n\t%2d times -
> Spend : %1f sec\n", i + 1, times[i]);
        sleep(2);
    }

    system("cls");

    for (int i = 0; i < TIMES; i++) {

```

```

        printf("%2d times -
> Spend : %lf sec\n", i + 1, times[i]);
        fprintf(output, "%2d times -
> Spend : %lf sec\n", i + 1, times[i]);
    }

    fclose(output);

    return 0;
}

```

二、選擇排序 (Selection sort)

```

# include <stdio.h>
# include <stdlib.h>
# include <unistd.h>
# include <windows.h>
# include <time.h>
# define TIMES 25
# define MAX 50000

int *generate_number (int *data) {
    for (int i = 0; i < MAX; i++) {
        data[i] = rand() % 1000000 + 1;
        ((i+1)%10 == 0) ? printf("%10d\n", data[i]) : printf("
%10d ", data[i]);
    }
    return (int *) data;
}

void check_number (int *data) {
    for (int i = 1; i < MAX; i++) {
        ((i+1)%10 == 0) ? printf("%10d\n", data[i]) : printf("
%10d ", data[i]);
    }
}

double selection_sort (int *data) {
    srand(time(NULL));
    clock_t t_start = clock();

```

```

        for (int i = 0; i < MAX; i++) {
            for (int j = i + 1; j < MAX; j++) {
                if (data[i] > data[j]) {
                    int temp = data[i];
                    data[i] = data[j];
                    data[j] = temp;
                }
            }
        }
    }

    clock_t t_end = clock();
    return ((double)t_end - t_start) / CLOCKS_PER_SEC;
}

int main () {
    int data[MAX] = {0};
    double times[TIMES] = {0};

    FILE *output = fopen("selection_sort_output.txt", "w");

    for (int i = 0; i < TIMES; i++) {
        generate_number(data);
        printf("\n\t%2d times Sorting Start...\n\n\n", i + 1);

        times[i] = selection_sort(data);
        sleep(1);

        printf("\n\t%2d times Sorting End...\n\n\n", i + 1);

        check_number(data);

        printf("\n\t%2d times -
> Spend : %lf sec\n", i + 1, times[i]);
        sleep(2);
    }

    system("cls");

    for (int i = 0; i < TIMES; i++) {

```

```

        printf("%2d times -
> Spend : %lf sec\n", i + 1, times[i]);
        fprintf(output, "%2d times -
> Spend : %lf sec\n", i + 1, times[i]);
    }

    fclose(output);

    return 0;
}

```

三、插入排序 (Insertion sort)

```

# include <stdio.h>
# include <stdlib.h>
# include <unistd.h>
# include <windows.h>
# include <time.h>
# define TIMES 25
# define MAX 50000

int *generate_number (int *data) {
    for (int i = 0; i < MAX; i++) {
        data[i] = rand() % 1000000 + 1;
        ((i+1)%10 == 0) ? printf("%10d\n", data[i]) : printf("
%10d ", data[i]);
    }
    return (int *) data;
}

void check_number (int *data) {
    for (int i = 1; i < MAX; i++) {
        ((i+1)%10 == 0) ? printf("%10d\n", data[i]) : printf("
%10d ", data[i]);
    }
}

double insertion_sort (int *data) {
    srand(time(NULL));
    clock_t t_start = clock();

```

```

        for (int i = 1; i < MAX; i++) {
            int insertion_index = i;
            while (insertion_index > 0 && (data[insertion_index-1] > data[insertion_index])) {
                int temp = data[insertion_index];
                data[insertion_index] = data[insertion_index-1];
                data[insertion_index-1] = temp;
                insertion_index -= 1;
            }
        }

        clock_t t_end = clock();
        return ((double)t_end - t_start) / CLOCKS_PER_SEC;
    }

int main () {
    int data[MAX] = {0};
    double times[TIMES] = {0};

    FILE *output = fopen("insertion_sort_output.txt", "w");

    for (int i = 0; i < TIMES; i++) {
        generate_number(data);
        printf("\n\t%2d times Sorting Start...\n\n\n", i + 1);

        times[i] = insertion_sort(data);
        sleep(1);

        printf("\n\t%2d times Sorting End...\n\n\n", i + 1);
        check_number(data);

        printf("\n\t%2d times -
> Spend : %lf sec\n", i + 1, times[i]);
        sleep(2);
    }

    system("cls");

    for (int i = 0; i < TIMES; i++) {

```



```

        printf("%2d times -
> Spend : %lf sec\n", i + 1, times[i]);
        fprintf(output, "%2d times -
> Spend : %lf sec\n", i + 1, times[i]);
    }

    fclose(output);

    return 0;
}

```

四、快速排序 (Quick sort)

```

# include <stdio.h>
# include <stdlib.h>
# include <unistd.h>
# include <windows.h>
# include <time.h>
# define TIMES 25
# define MAX 50000

int *generate_number (int *data) {
    for (int i = 0; i < MAX; i++) {
        data[i] = rand() % 1000000 + 1;
        ((i+1)%10 == 0) ? printf("%10d\n", data[i]) : printf("
%10d ", data[i]);
    }
    return (int *) data;
}

void check_number (int *data) {
    for (int i = 1; i < MAX; i++) {
        ((i+1)%10 == 0) ? printf("%10d\n", data[i]) : printf("
%10d ", data[i]);
    }
}

int Partition(int *data, int front, int end){
    int i = front - 1, temp = 0;
    for (int j = front; j < end; j++) {
        if (data[j] < data[end]) {

```

```

        i++;
        temp = data[i];
        data[i] = data[j];
        data[j] = temp;
    }
}
i++;
temp = data[i];
data[i] = data[end];
data[end] = temp;
return i;
}

double quick_sort (int *data, int front, int end) {
    srand(time(NULL));
    clock_t t_start = clock();

    if (front < end) {
        int seed = Partition(data, front, end);
        quick_sort(data, front, seed - 1);
        quick_sort(data, seed + 1, end);
    }

    clock_t t_end = clock();
    return ((double)t_end - t_start) / CLOCKS_PER_SEC;
}

int main () {
    int data[MAX] = {0};
    double times[TIMES] = {0};

    FILE *output = fopen("quick_sort_output.txt", "w");

    for (int i = 0; i < TIMES; i++) {
        generate_number(data);
        printf("\n\t%2d times Sorting Start...\n\n\n", i + 1);

        times[i] = quick_sort(data, 0, MAX-1);
        sleep(1);
    }
}

```

```

        printf("\n\t%2d times Sorting End...\n\n\n", i + 1);

        check_number(data);

        printf("\n\t%2d times -
> Spend : %lf sec\n", i + 1, times[i]);
        sleep(2);
    }

    system("cls");

    for (int i = 0; i < TIMES; i++) {
        printf("%2d times -
> Spend : %lf sec\n", i + 1, times[i]);
        fprintf(output, "%2d times -
> Spend : %lf sec\n", i + 1, times[i]);
    }

    fclose(output);

    return 0;
}

```

五、堆積排序 (Heap sort)

```

# include <stdio.h>
# include <stdlib.h>
# include <unistd.h>
# include <windows.h>
# include <time.h>
# define TIMES 25
# define MAX 50000

int *generate_number (int *data) {
    for (int i = 1; i < MAX; i++) {
        data[i] = rand() % 1000000 + 1;
        ((i+1)%10 == 0) ? printf("%10d\n", data[i]) : printf("
%10d ", data[i]);
    }
    return (int *) data;
}

```

```

void check_number (int *data) {
    for (int i = 1; i < MAX; i++) {
        ((i+1)%10 == 0) ? printf("%10d\n", data[i]) : printf("
%10d ", data[i]);
    }
}

void MaxHeapify(int *data, int root, int length) {
    int left_child = 2 * root;
    int right_child = 2 * root + 1;
    int largest = 0;

    if (left_child <= length && data[left_child] > data[root])
    {
        largest = left_child;
    } else {
        largest = root;
    }

    if (right_child <= length && data[right_child] > data[larg
est]) {
        largest = right_child;
    }

    if (largest != root) {
        int temp = data[largest];
        data[largest] = data[root];
        data[root] = temp;
        MaxHeapify(data, largest, length);
    }
}

void BuildMaxHeap (int *data) {
    for (int i = MAX/2; i >= 1; i--) {
        MaxHeapify(data, i, MAX-1);
    }
}

double heap_sort (int *data) {

```

```

    srand(time(NULL));
    clock_t t_start = clock();

    BuildMaxHeap(data);

    int size = MAX-1;
    for (int i = size; i>=2 ; i--) {
        int temp = data[1];
        data[1] = data[i];
        data[i] = temp;
        size--;
        MaxHeapify(data, 1, size);
    }

    clock_t t_end = clock();
    return ((double)t_end - t_start) / CLOCKS_PER_SEC;
}

int main () {
    int data[MAX] = {0};
    double times[TIMES] = {0};

    FILE *output = fopen("heap_sort_output.txt", "w");

    for (int i = 0; i < TIMES; i++) {
        generate_number(data);
        printf("\n\t%2d times Sorting Start...\n\n\n", i + 1);

        times[i] = heap_sort(data);
        sleep(1);

        printf("\n\t%2d times Sorting End...\n\n\n", i + 1);

        check_number(data);

        printf("\n\t%2d times -
> Spend : %1f sec\n", i + 1, times[i]);
        sleep(2);
    }

```

```

system("cls");

for (int i = 0; i < TIMES; i++) {
    printf("%2d times -
> Spend : %lf sec\n", i + 1, times[i]);
    fprintf(output, "%2d times -
> Spend : %lf sec\n", i + 1, times[i]);
}

fclose(output);

return 0;
}

```

參、執行結果

```

20411 19617 9048 10920 22348 6872 58 23218 7016 22810
4084 3395 6830 24987 442 15195 5745 11121 32536 31706
20318 9382 28100 20025 30047 29324 32406 32120 20160 3152
11733 12119 23185 25554 6324 29641 29259 4276 19484 26790
11958 12722 28844 17824 5129 30769 30719 25720 18771 4634
12077 6514 24347 26088 23188 28678 26947 21760 16088 8152
28723 30785 15144 19946 32051 2781 22723 28176 17943 30567
29627 22693 15394 15149 24825 26749 549 5063 13698 13594
3559 10992 4857 15651 30921 13461 7524 5393 17322 13100
32307 28204 4617 22087 6270 2486 8297 10089 21134 27750
31226 31000 6303 26136 12661 19507 31972 16834 8520 8070
6348 12714 30451 23964 4996 19414 3506 23764 29117 10948
23644 9655 4783 1628 31371 29577 32349 22819 253 8990
26907 26033 18746 1008 4174 12389 20599 16238 21620 16727
16827 19094 13412 17038 27456 5573 3542 20084 8988 28366
18213 646 24352 18762 18004 10832 1025 24880 22864 5785
12213 27558 29556 29915 800 18966 25682 15094 12016 29571
27569 31524 18374 2078 20178 13360 9622 17172 21240 29155
21072 16043 16678 20928 19392 16013 22315 16369 7347 29795

1 times Sorting Start...

1 times Sorting End...

1 times -> Spend : 0.099000 sec

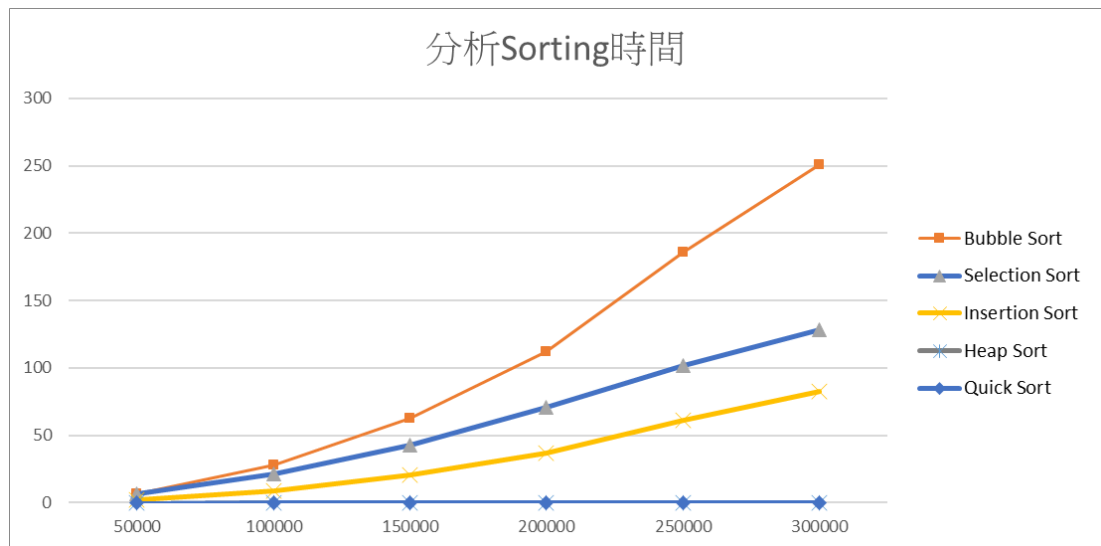
```

```

1 times -> Spend : 0.099000 sec
2 times -> Spend : 0.100000 sec
3 times -> Spend : 0.102000 sec
4 times -> Spend : 0.100000 sec
5 times -> Spend : 0.093000 sec
6 times -> Spend : 0.160000 sec
7 times -> Spend : 0.100000 sec
8 times -> Spend : 0.099000 sec
9 times -> Spend : 0.100000 sec
10 times -> Spend : 0.101000 sec
11 times -> Spend : 0.092000 sec
12 times -> Spend : 0.101000 sec
13 times -> Spend : 0.100000 sec
14 times -> Spend : 0.090000 sec
15 times -> Spend : 0.150000 sec
16 times -> Spend : 0.099000 sec
17 times -> Spend : 0.106000 sec
18 times -> Spend : 0.100000 sec
19 times -> Spend : 0.100000 sec
20 times -> Spend : 0.100000 sec
21 times -> Spend : 0.097000 sec
22 times -> Spend : 0.089000 sec
23 times -> Spend : 0.090000 sec
24 times -> Spend : 0.185000 sec
25 times -> Spend : 0.126000 sec

-----
Process exited after 693.4 seconds with return value 0
請按任意鍵繼續 . . .

```



肆、 討論

時間上來說，Bubble sort 真的最花時間，然後 Quick sort 跟 Heap sort 議整個快到不行，在測資 500000 的時候最有感覺。以下全部花費時間 Bubble Sort > Selection Sort > Insertion Sort > Quick Sort ≈ Heap Sort。

伍、 心得

我寫好以後發現他真的要跑好久，然後一開始我沒有想好，沒有寫任何可以寫檔記錄的功能，然後分開寫的五隻程式，每隻跑六次，我花很多時間在等待要執行下一次，然後還會跑完沒記錄到，然後後來搞了寫檔，又寫了一隻批次檔，可以幫我跑全部，結果我寫檔的部分因為都是複製貼上寫好的同個模板，居然把 Selection Sort 跟 Bubble Sort 的輸出寫到同檔案，後來要寫報告的時候才發現大事不妙，只好再跑一次，然後再重新等那 500000 的跑完，整個氣到不行。