Project 1 Report

Implementation and Performance Evaluation of Different Sorting Algorithm

姓名：謝晉維

b08209006@ntu.edu.tw

團隊其他成員: 無

所有 Source Code 皆可以在我的 github 找到。

如果要測試檔案可以直接到 bin 資料夾底下執行 shell script 比較快

若想要測試每個執行檔也可以先在 Final Report of Sorting 資料夾輸入 make（也可以先 make clean），再去 bin 資料夾底下執行./Sorting，執行如果不符合要求會跳出提示，按照提示輸入即可。

（整個 project 的結構在 github 的 readme 中可找到）

1. Selected Sorting Algorithm

在 src/Sorting_tool.cpp 中可看到實作的三種 Sorting 方法

1. Selection Sort: 從未排序好的 data 中找到最小值後，丟到未排序好的最左邊，成為已排序好的 data，重複前述步驟直到所以資料排序完成。

2. Insertion Sort: 從未排序的 data 中選一個並且往前排排到適當的位置後成為以排序的 data，重複前述步驟直到所以資料排序完成。

3. Bubble Sort: 兩兩比較 data，當發現順序不對就交換，走完全部 data 即排序完成。

4. Heap Sort: 將資料轉為 max-heap 的形式，逐步取出最大值與最後一個元素交換再縮小範圍重複取出最大值交換的步驟直到只剩一筆未排序資料。

2. List the API of each sorting algorithm

1. Selection Sort:

```
void SortingTool::selectionsort(void *const base, size_t num, size_t size, int (*compar) (const void*, const void*))
```

2. Insertion Sort:

```
void SortingTool::insertionsort(void *const base, size_t num, size_t size, int (*compar) (const void*, const void*))
```

3. Bubble Sort:

void SortingTool::bubblesort(void *const base, size_t num, size_t size, int (*compar) (const void*, const void*))

4. Heap Sort:

void SortingTool::heapsort(void *const base, size_t num, size_t size, int (*compar) (const void*, const void*))

3. Implement or use sorting algorithm with Schwartzian transform

利用 index 進行排序，排序好後用此 index output 出檔案。

實作此方法的好處是可以省下做 memory copy 的時間。

實作 code 在 src/main.cpp，測試方法可以直接到 bin 資料夾底下執行寫好的 shell script（SS_SortingComplexSch.sh、IS_SortingComplexSch.sh、BS_SortingComplexSch.sh、HS_SortingComplexSch.sh）

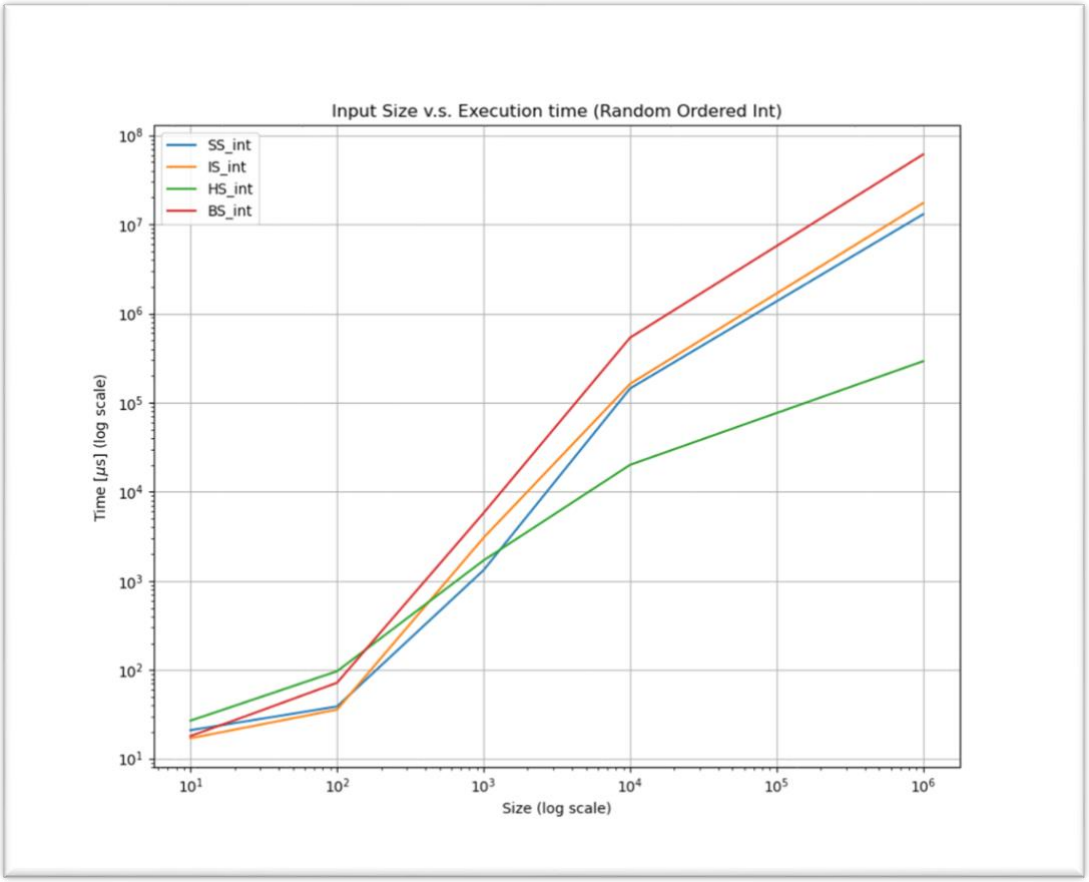
此題的實作 input data 為隨機產生的 complex data（複雜資料比較有差），如果想要產生不同的 data 可以到 bin 資料夾底下執行 ComplexDataGenerate.sh 或是執行./ComplexDataGenerate <數目>。

4. Performance Evaluation

以下的圖為部分的執行結果做圖，若要看每個資料在每個 sort 的執行結果可以在 Final Report of Sorting/outputs 底下的圖看，或是執行 Final Report of Sorting/bin 下的 shell script 看結果。

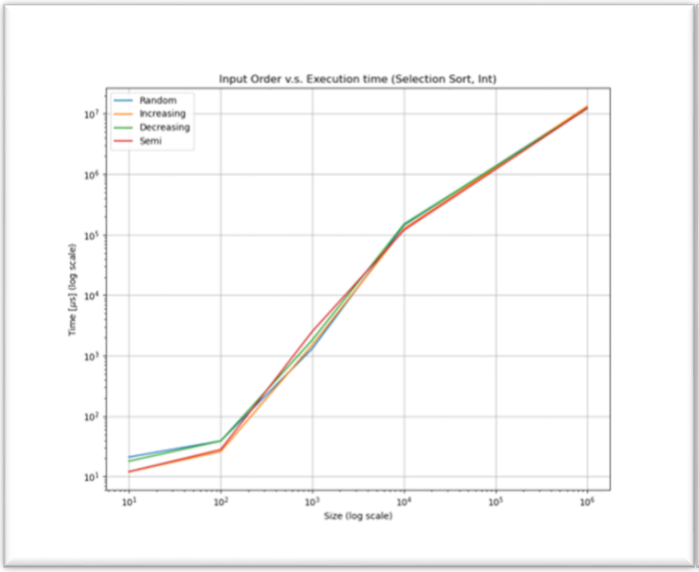
Project 架構圖

1.1. Input size (from small to large data set) vs Execution Time

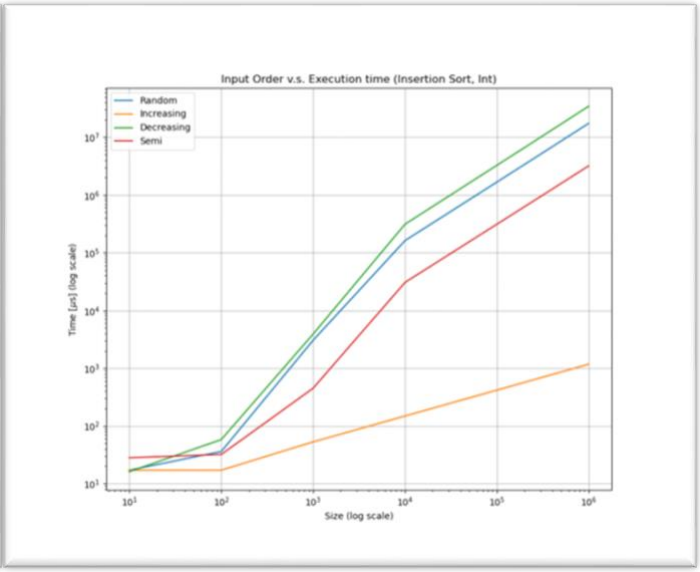


1.2. Input order (increasing order, decreasing order, semi-ordered, random) vs Execution Time

下圖為 SS，排列資料為整數

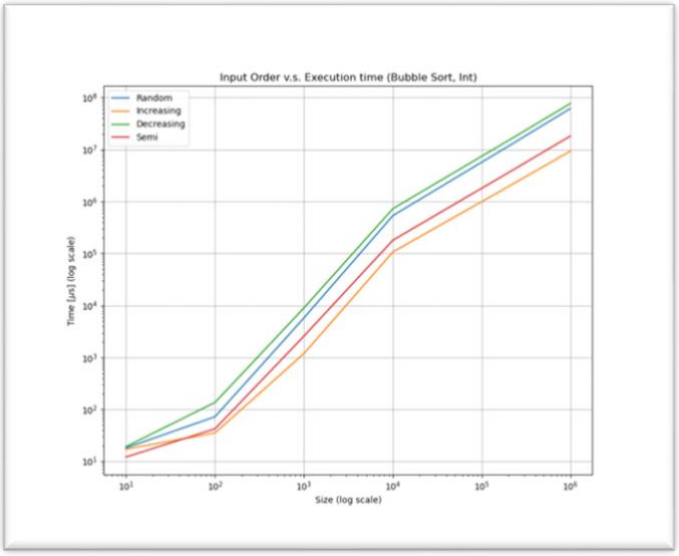


下圖為 IS，排列資料為整數

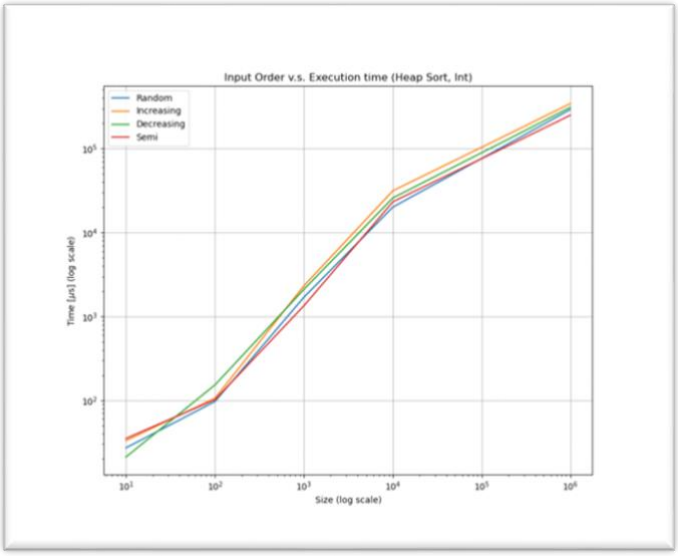


.

下圖為 BS，排列資料為整數



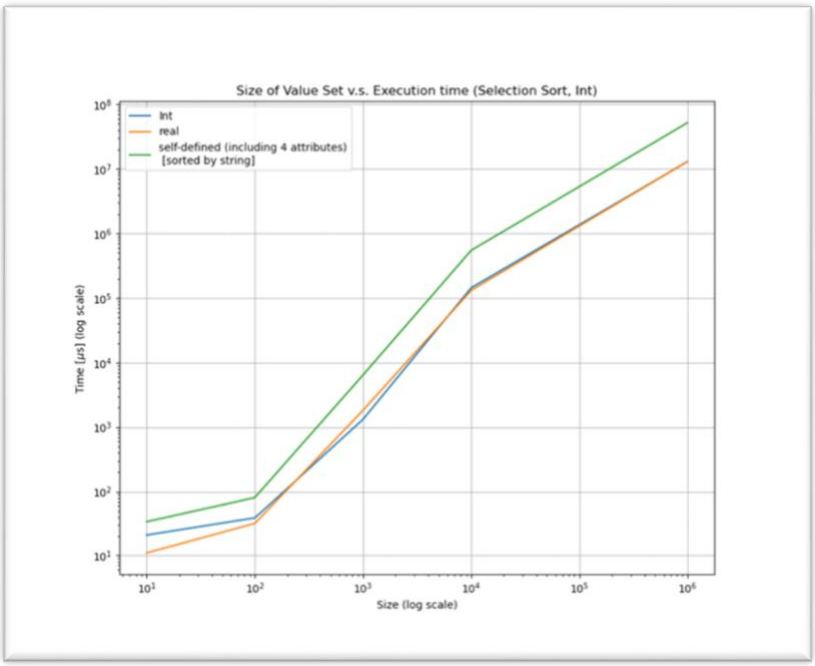
下圖為 HS，排列資料為整數



1.3. The size of value set (real value, integer value, category value(small, large)) vs Execution Time

1.4. Complex data structure/Simple data vs Execution Time

（此兩題畫在同一張圖中）

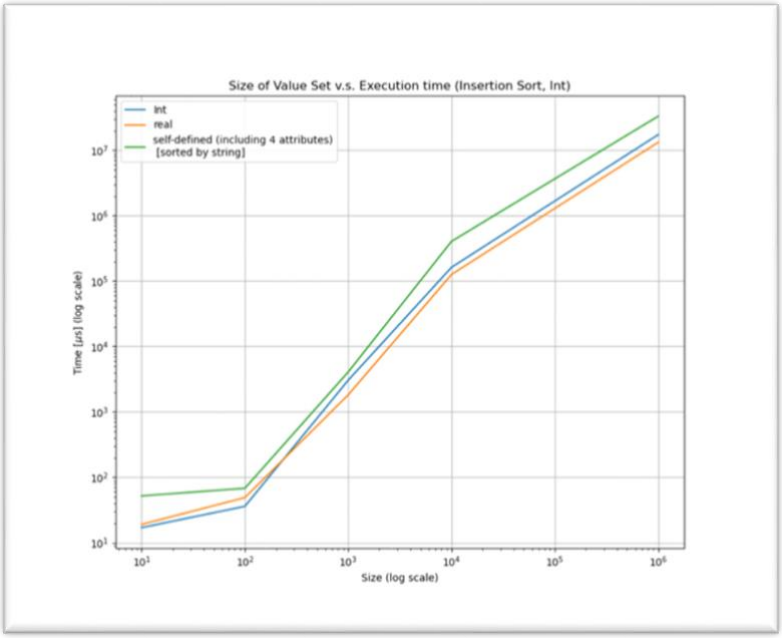


左圖為 SS 三種類型資料的排列

（int, double, 自訂 data）

自訂 data 中有：

firstname, lastname, ID, score

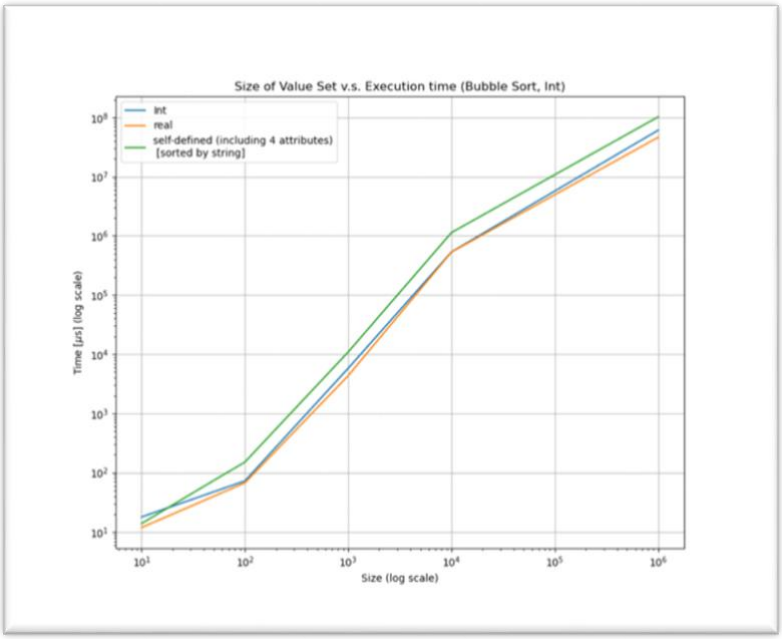


左圖為 IS 三種類型資料的排列

（int, double, 自訂 data）

自訂 data 中有：

firstname, lastname, ID, score

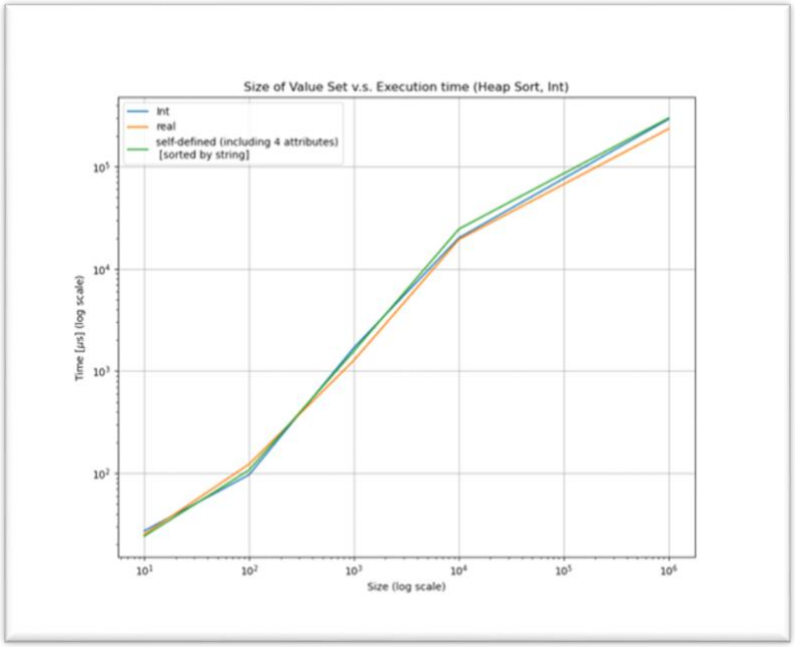


左圖為 IS 三種類型資料的排列

（int, double, 自訂 data）

自訂 data 中有：

firstname, lastname, ID, score



左圖為HS 三種類型資料的排列

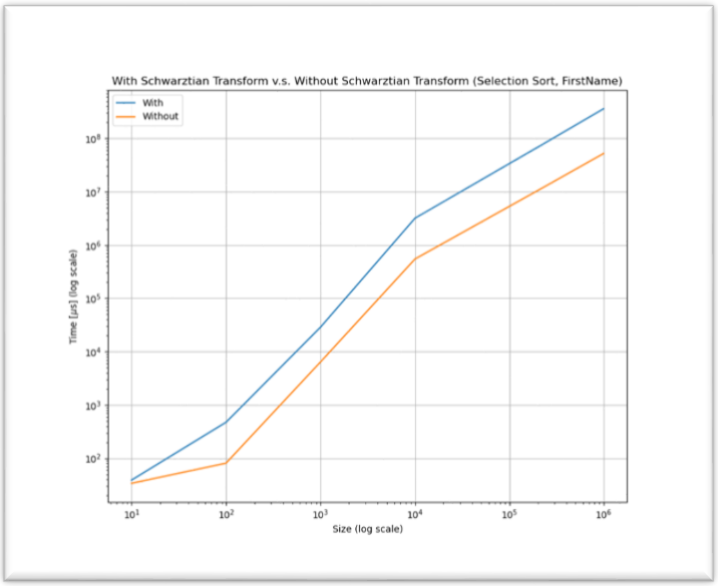
（int, double, 自訂 data）

自訂 data 中有：

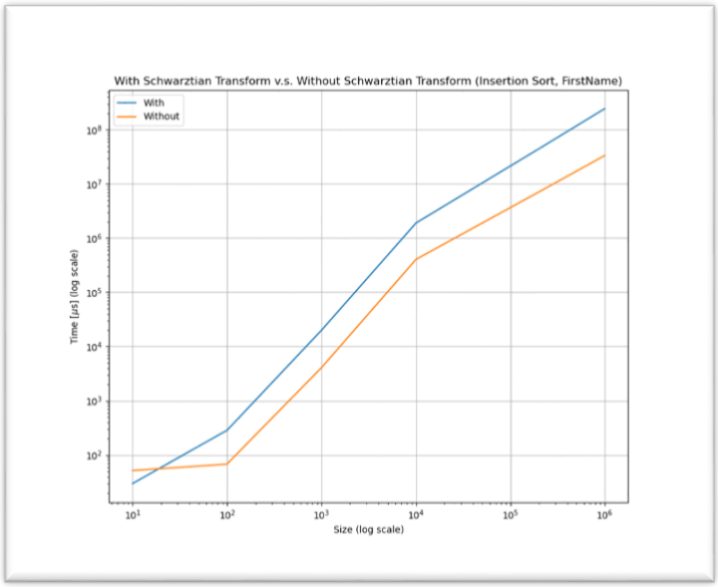
firstname, lastname, ID, score

1.5. With or without Schwartzian transform vs Execution Time

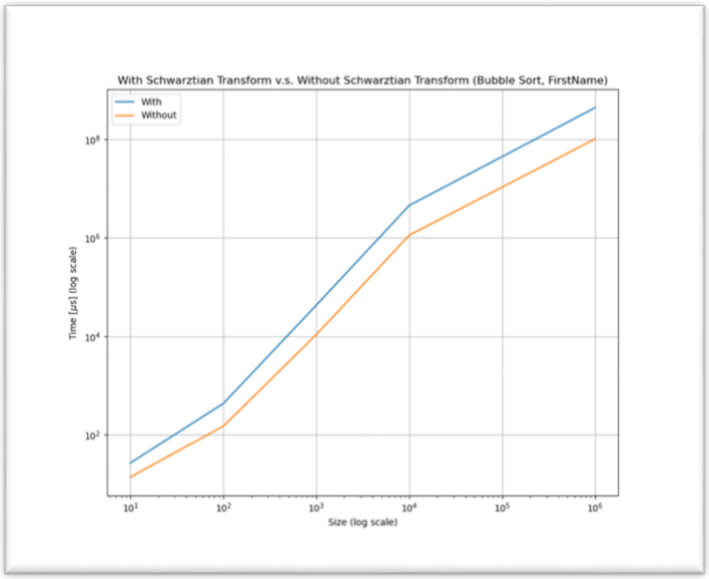
Selection Sort (sorted by FirstName)



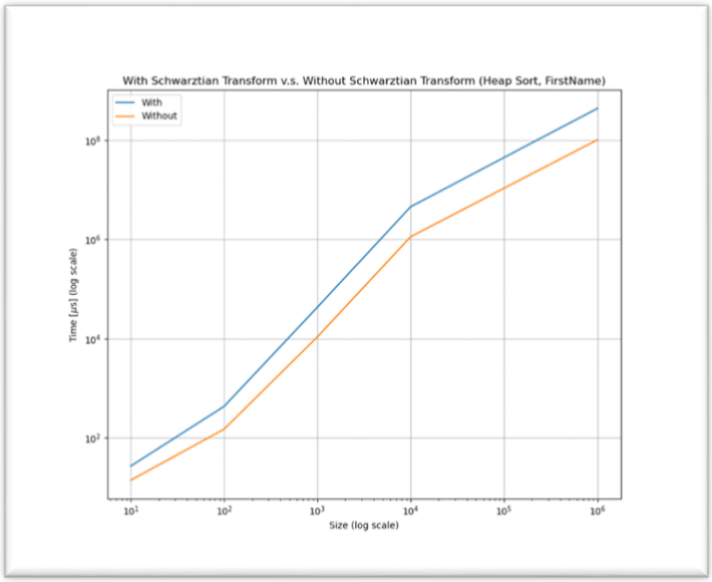
Insertion Sort (sorted by FirstName)



Bubble Sort (sorted by FirstName)



Heap Sort (sorted by FirstName)



5. Discussion

從第四題的表現中可看出

1. Input Size v.s. Execution Time

當 input size 越大執行時間越大，且 SS, IS, BS 複雜度為$O(N^2)$（圖的座標都有取 log），圖為線性的，因此可得知此訊息，而 HS 的複雜度為$O(NlogN)$。

2. Input Order v.s. Execution Time

(a) SS 中，不論是 random, increasing, decreasing, semi ordered 的 input，所需時間都差不多，這也可從演算法中理解，因為每次都要跑過還沒排列的矩陣找最小值，因此不管怎樣的順序都沒什麼影響。

(b) IS 中所需時間，decreasing > random > semi > increasing，且 increasing 的時間幾乎為$O(N)$（semi 也差不多只是要多多少就看 semi 的程度）。從演算法層面理解，因為 IS 是在未排列好的數字中一個一個找看需不需要往前排，要的話就一直讓此數字往前，increasing order 已經排列好了，所以只需要檢查過整個矩陣就可以排列完成。

(c) BS 中所需時間 decreasing > random > semi > increasing，但時間複雜度差不多。從演算法理解，每次都要兩兩比較，只是 increasing 可以少掉交換的步驟因此會快一些，但還是要檢查n^2次。

(d) HS 中 random, increasing, decreasing, semi ordered 所需時間差不多。從演算法理解每次找出最大值都是從 max-heap 中找，根據建立 heap 的方式可能會花不同時間但複雜度一樣。

3 and 4.

(a) 大的 size of value set 會花比較多時間排序，基本上應該是需要越多儲存空間的資料需要越多時間，因為需要複製及交換的時間都比較長。而在圖中 double 比 int 快的可能原因是資料本身的值的問題（原始排列程度）可能會有影響。

(b) 越複雜的資料所需要的儲存空間越多，在執行上就需要花更多時間，即使都是$\Theta(1)$但也有常數上的差別。

5. Schwartzian Transform 的結果在同個 size 下花的時間比沒有做的還對，跟理論上不太符合，因為 Schwartzian Transform 少了很多 memory copy 的行為，理論上應該要比沒有實作快不少。推測造成此現象的原因是我在實作 Schwartzian Transform 時 comparsion function 不是寫在 SortingTool 的 class 當中，調用函式的計算的時候花了更多的時間導致此現象。

6. Conclusion

實作上的執行時間，基本上跟演算法中理論的時間複雜度差不多。

1. 在 SS、BS、IS 中複雜度為$O(N^2)$，HS 中複雜度為$O(NlogN)$。
- SS 的 best case、worst case 都差不多為$O(N^2)$。
- IS 會有 Best Case 是$O(N)$、worst case 是$O(N^2)$、Average case 是$O(N^2)$。
- BS 在 Best, Worst, Average 都是$O(N^2)$在不同 case 上會有常數的差別。

- HS 在 Best, Worst, Average 都是$O(NlogN)$在不同 case 上會有常數的差別。

2. 要排列的資料單一值所需儲存空間越大，所需排列時間越長。即演算法的複雜度不變，但常數有差。

3. Schwartzian Transform 在此次實作中花的時間比沒有實作 Schwartzian Transform 多，與理論中不符合，可能原因已在討論中分析。

7. References

無