**一、Contains**

本次實驗共實作了 18 個程式，分為三大類排序方法：Merge Sort、Quick Sort 與內建的 std::sort。其中 Merge Sort 與 Quick Sort 各實作了 4 種不同終止邊界的版本，以及額外 3 個針對特殊資料情況（元素皆相同、有序、反向有序）的測試；std::sort 則實作 4 種資料類型的測試。

**二、Terminal Conditions**

在 Merge Sort 與 Quick Sort 中，我設定了四種子問題大小終止邊界（10、20、50、100），當子問題的資料筆數小於該邊界時，即停止遞迴並改用非遞迴的排序方法。對於特殊資料類型（元素皆相同、有序、反向有序），則固定採用邊界值 100 進行測試。

各程式對應如下：

* Merge Sort：merge1.cpp（邊界 10）、merge2.cpp（20）、merge3.cpp（50）、merge4.cpp（100）、特殊資料：merge元素同.cpp、merge反向.cpp、merge有序.cpp
* Quick Sort：quick1.cpp（邊界 10）、quick2.cpp（20）、quick3.cpp（50）、quick4.cpp（100）、特殊資料：quick元素同.cpp、quick反向.cpp、quick有序.cpp
* std::sort 測試：stdsort.cpp（隨機）、stdsort元素同.cpp、stdsort反向.cpp、stdsort有序.cpp

**三、非遞迴排序方法（Non-Recursive Method）**

在子問題規模小於設定邊界值時，Merge Sort 與 Quick Sort 均改用 Bubble Sort來處理。雖然 Bubble Sort 時間複雜度為 $O(n^2)$，但由於其簡單易實作、且僅應用於資料筆數很少的情況，對整體效能影響可控。

**四、Different Input Sizes**  
對於邊界值測試，測試資料筆數分別為：1,000、10,000、100,000、1,000,000。對於特殊資料測試（元素皆同、有序、反向），則固定測試 1,000,000 筆資料，邊界值固定為 100。

透過不同演算法、終止條件與資料特性的組合，可以觀察到各種情況下排序效能的差異。

**五、Experiment Results**

| 演算法 | 邊界 | 資料量 | 花費時間 |
| --- | --- | --- | --- |
| Merge Sort | 10 | 1000000 | 350090μs |
| Merge Sort | 20 | 1000000 | 334904μs |
| Merge Sort | 50 | 1000000 | 350035μs |
| Merge Sort | 100 | 1000000 | 412341μs |
| Quick Sort | 10 | 1000000 | 200075μs |
| Quick Sort | 20 | 1000000 | 184949μs |
| Quick Sort | 50 | 1000000 | 166646μs |
| Quick Sort | 100 | 1000000 | 261666μs |

**六、Try**

測試條件：資料筆數皆為 1,000,000，邊界值為 100。

| 資料型態 | Merge Sort | Quick Sort | std::sort |
| --- | --- | --- | --- |
| 隨機 | 412341μs | 261666μs | 164141μs |
| 有序 | 236897μs | 卡死 | 70377μs |
| 反向 | 215109μs | 卡死 | 84113μs |
| 元素相同 | 219591μs | 卡死 | 139511μs |

有序資料是先使用 std::sort(arr.begin(), arr.end()) 產生。

反向資料是有序資料再用 sort(arr.begin(), arr.end(), greater<int>()) 處理。

元素相同資料是全部設為相同值（如 arr[i] = 100）。

**補充說明：**

在 Quick Sort 的原始實作中，pivot（基準點）選擇的是區間最後一個元素。當資料為有序、反向、有大量重複元素時，會導致每次 partition 產生極端不平衡的分割，退化為時間複雜度 O(n²)，進而造成程式效能急劇下降甚至「卡死」。改進方法：可以考慮改用中位數作為 pivot，

例如選擇 (arr[left] + arr[right]) / 2 。